

図解

マイクロコンピュータ

64180

の使い方

Ph.D. 石川 知雄 編 ● オーム社



図解マイクロコンピュータ

Z-80の使い方

(A 5判・200頁)

横田英一 著

本書は、インテル8080とともに広く使われているZ-80の使い方について、CPUの概要から各種命令、動作、プログラムのテクニックに至るまでを初学者でも十分理解できるよう平易に解説しました。

図解マイクロコンピュータ

Z-80アセンブラプログラミング入門

(A 5判・236頁)

湯田幸八／伊藤 彰 共著

本書は、自分なりに考えて工夫したツールをもとに、制御技術の基本を理解し、手軽にアプリケーションができるように、製作と機械語によるプログラミングを通じて、体験的に学べるようにまとめたもので、専門知識がなくてもわかるように解説してあります。

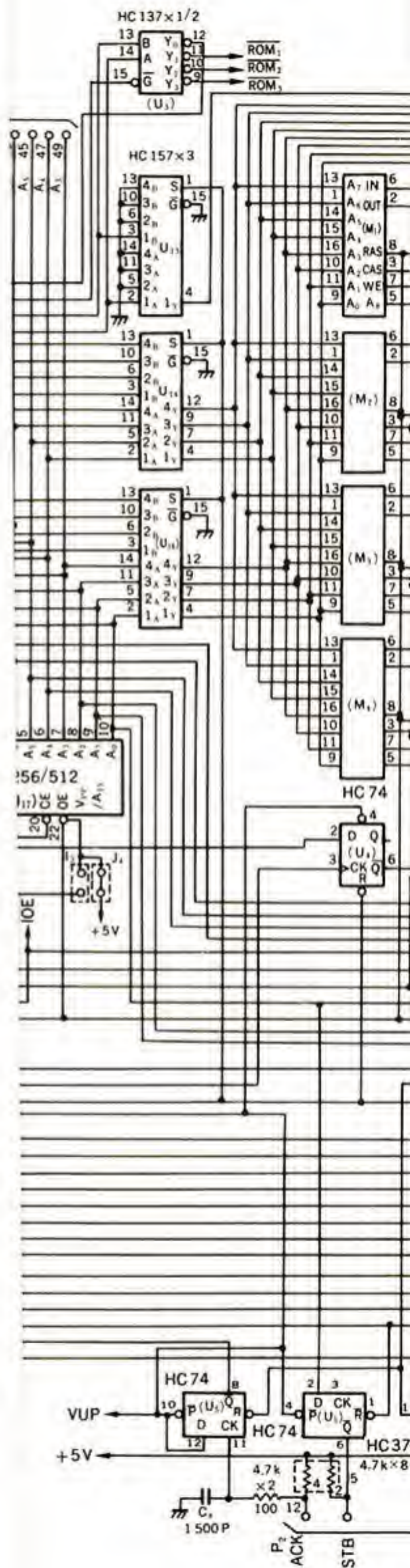
図解32ビットマイクロコンピュータ

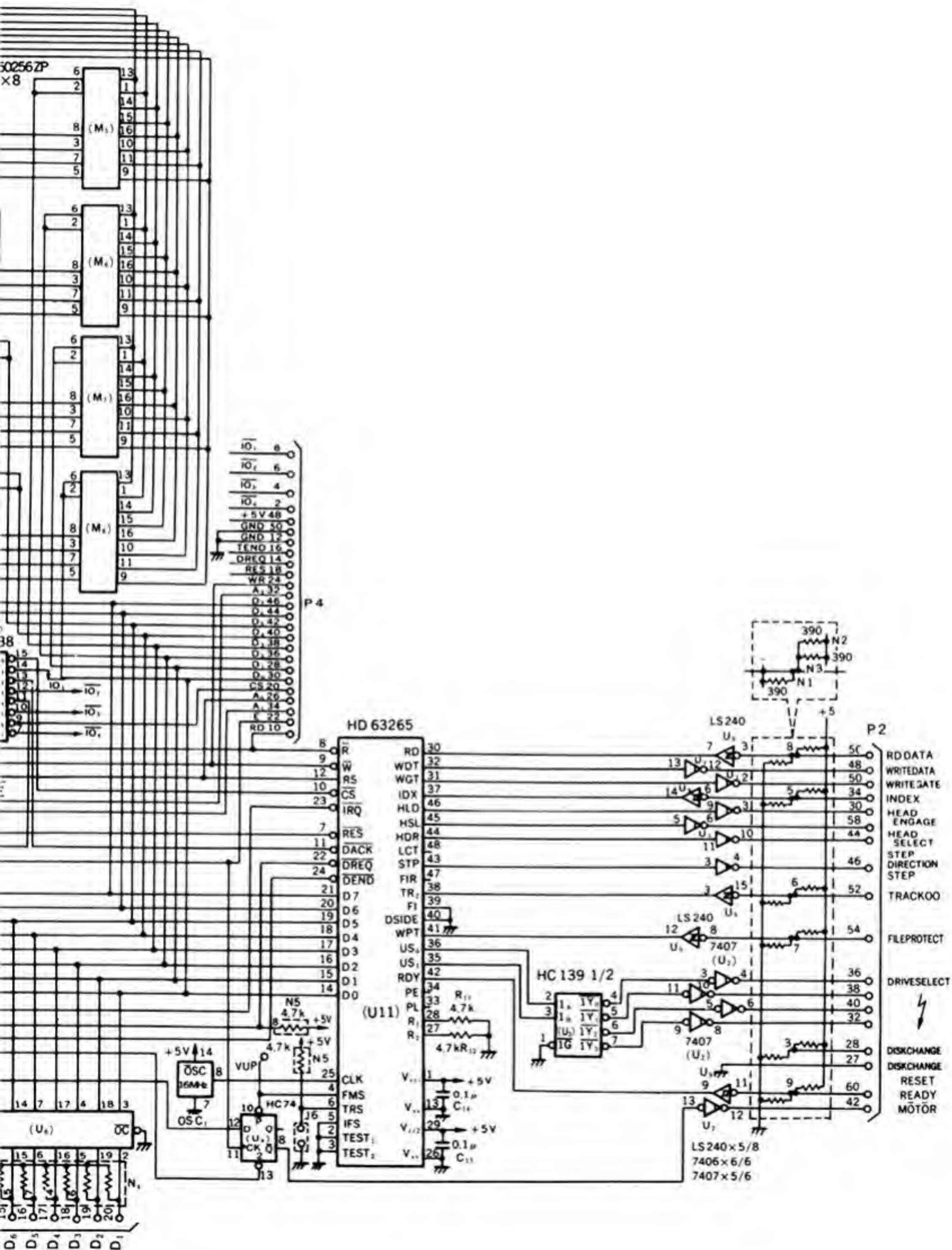
80386の使い方

(A 5判・218頁)

W.B. スルヤント 著

本書は、32ビットマイクロプロセッサ80386の特長を生かすための基本的機能と命令を中心に、項目ごとにまとめたもので、80386に加わった新しい概念を8086の延長線上にとらえて解説しているので、8086ユーザーに大変理解しやすくなっています。





図解

マイクロコンピュータ

64180

の使い方

● Ph.D. 石川 知雄 編

オーム社

執筆者一覧

石川 知雄	株式会社日立製作所
馬場 志郎	株式会社日立製作所
川下 智恵	株式会社日立製作所
越路 誠一	株式会社日立製作所
北川 信男	株式会社日立製作所
高野 誠	日立マイクロコンピュータエンジニアリング株式会社
井戸 慎一	日立マイクロコンピュータエンジニアリング株式会社
星 恭彦	日立マイクロコンピュータエンジニアリング株式会社

- CP/M, CP/M-86, CP/M-Plus, CP/M-68KはDigital Research社の商標.
- MS-DOSはMicrosoft社の商標.
- MBASICはMicrosoft社の商標.
- WordstarはMicroPro社の商標.
- dBASEはAshton-Tate社の商標.

本書は、「著作権法」によって、著作権等の権利が保護されている著作物です。

本書の全部または一部につき、無断で次に示す〔 〕内のような使い方をされると、著作権等の権利侵害となる場合がありますので御注意ください。

〔転載、複写機等による複写複製、電子的装置への入力等〕

学校・企業・団体等において、上記のような使い方をされる場合には特に御注意ください。

お問合せは下記へお願いします。

〒101 東京都千代田区神田錦町3-1 Tel.03-233-0641

株式会社 オーム社出版局（著作権担当）

推薦のことば

過 去

1970 年度初頭のマイコンの登場は、世の中に大きな変化をもたらしました。以前は夢であった真のパーソナルコンピュータが現実のものとなったのです。

その頃、既存の大きなコンピュータ・メーカーはデスクトップ・コンピューティングなる概念には簡単にはなじめなかったのです。このことは将来の夢に駆られた数多くの小さなハードウェア、ソフトウェアの会社に対し、大きなオープン・マーケットを提供することになったのです。

私の最初のパソコンは 8080 A CPU ボード (2 MHz) と 4 K バイト RAM ボードをのせた IMSAI のボックスでした。プログラミングはバイナリコードを前面のパネルスイッチから入力することで行い、アプリケーションも入出力の 8 個のスイッチと 8 個の LED 表示で限定されたものでした。

このようなみすばらしい始まりでしたが、新しい技術により性能が急速に向上するまでにはあまり長い時間を要しませんでした。強化された Z80 プロセッサ、フロッピーディスク、集積度の高い RAM、低価格のビデオ・ターミナルの登場が、CP/M (Digital Research)、MBASIC (Microsoft)、Wordstar (MicroPro)、dBASE (Ashton-Tate) などのパイオニアとなったソフトウェア製品に、格好の土台を提供したのです。これらの、アセンブリ言語で書かれ、人手で高速化、効率化のためチューン・アップされたプログラムは、その後のテスト期間に耐えたのです。事実これらのプログラムは現在の水準に対しても高度のプログラム芸術といえましょう。

その後も技術の進歩は続き、1970 年代の末にはインテル 8088/8086、モトローラ 68000、Zilog Z8000 の 16 ビットの戦いが焦点となりました。新しいマイコンに人材が集中し、8 ビットの市場は放棄されたかに見えました。実際、1980 年代の初めに IBM PC と Apple Lisa が登場し、PC が 16 ビット (今日では 32 ビット) へ移ることが明らかとなったのです。

推 薦 の こ と ば

しかしながら数百万台の8ビットマイコン、特にZ80は引続きプリンタ、ターミナル、データ・コミュニケーション、産業制御などの分野で使用されています。これらの分野の人達が望んでいるのは、16ビットではなくて下記の3つの目標を満たす新しい8ビット・プロセッサなのです。

- 高性能化と高集積化…性能面、コスト面のキーとなる項目を満たすこと、たとえば64Kバイトのリミットを破ること、部品点数が少なくなることなど。
- フル・コンパチビリティ…数百万ステップのソフトウェア投資が守れること、プログラマの訓練も不要なこと。
- CMOS プロセス…高速、低消費電力のシステム向けのプロセスとして。

現 在

この本を読まれるにつれ、皆様は64180が上記の目標を巧みに満たしていることに気付かれるでしょう。実際、64180は消え去るのではなく、長い、健全な寿命をもつ8ビットのマーケットを再認識させたのです。ここアメリカでは、64180はCBBS (Computer Bulletin Board System: コンピュータ化掲示板) からホーム・セキュリティ・システムまでのあらゆるシステムに、時にはレース・カー用のオンボードコンピュータとして、使われています。64180のメリットをフルに生かした新しいハードウェア、ソフトウェアが毎日のように出ており、熱心なユーザ・グループの支持を得ています。

64180は①高性能、②コスト・パフォーマンス、③低消費電力—これらは大多数の設計者が望むところですが—を要求される応用に理想的です。また、アーキテクチャが素直で理解しやすく、応用ソフトの開発も容易です。また、無償のものも含めて数百（あるいは数千？）の既存アプリケーション・ソフトウェアと言語もあります。

将 来

明るいというにつきましょう。64180は今後とも伸びる数多くの製品の中心として、長期にわたり使用され続けるでしょう。さて、この本により皆様は自分自身の応用に対し64180を適用するための勉強を開始できます。

Microfuture 社 社長 トーマス W. カントレル

(編者和訳)

Recommendation

The Past

In the early '70s, the invention of the first 8-bit microprocessors changed the world forever. For the first time, the dream of a truly 'personal' computer became possible.

The existing 'big' computer companies couldn't easily adapt to the concept of desktop computing. This left the market open for hundreds of tiny hardware and software companies, all driven by a vision of the future.

One of my earliest PCs was an IMSAI box containing an 8080A CPU board (running at 2MHz) and a 4Kbyte RAM board. 'Programming' was accomplished by entering the binary opcodes on the front panel switches. Of course, the resulting 'application program' was limited by the available 8 switches and 8 LEDs !

From these humble beginnings, it wasn't long before new technology was able to boost performance. The arrival of the enhanced Z80 processor, floppy disks, denser RAM and low-cost video terminals provided the 'platform' for the pioneering software products like CP/M (Digital Research), MBASIC (Microsoft), Wordstar (MicroPro) and dBASE (Ashton-Tate). These programs, all written in assembly language and hand tuned for speed and efficiency, have withstood the test of time. Even by today's standards, they stand as shining examples of the art of programming.

Inevitably, technology marched on and by the end of the decade, focus had switched to the battle for 16-bit dominance between the Intel 8088/8086, Motorola 68000 and Zilog Z8000. With resources devoted to new micros, it seemed the 8-bit market was being abandoned. Indeed, the early '80s introduction of the IBM PC and Apple Lisa confirmed that 'PCs' would move onward to 16-bits (and today, 32-bits).

Recommendation

Yet, millions of 8-bit micros, especially the Zilog Z80, were still being used in applications such as printers, terminals, data communications, industrial control, etc. What these customers needed was not 16-bits but rather a new 8-bit chip fulfilling three key goals.

- **High Performance/Integration**...The CPU had to break the key performance and cost barriers such as 64Kbyte memory limit and excessive chip count.
- **Total Compatibility**...To maintain the investment of millions of lines of code (and the resulting programmer training!).
- **CMOS Process**...Recognizing the '80s best process technology for fast, low-power systems.

The Present

As you read this book, you'll find that the Hitachi HD64180 has met these goals admirably-indeed, the '180 has 'revived' the 8-bit market which, rather than disappearing, has a long, healthy life ahead.

Here in the US, the '180 is being used in everything from a CBBS (Computer Bulletin Board System) to a digital video home security system-even an on-board instrumentation computer for a race car! New hardware and software which takes full advantage of the '180 emerges daily and is backed by an active network of user groups.

The '180 is ideal for applications requiring good performance, efficient design and low power-features most designers appreciate. Development is easy because the architecture is straight-forward and understood by many. Hundreds (even thousands) of different applications and languages are available-many for free!

The Future

...is bright, and the '180 will continue to be the centerpiece of an ever growing variety of products for many years. Now, with this book, you can start learning how to put the '180 to work in your own application.

Thomas W. Cantrell
President, Microfuture

は し が き

1971 年、マイコンが最初に世の中に出てから十余年経過し、マイコンの応用は広い分野に定着し、またマイコン LSI 自体も第 3、第 4 世代へと着実な発展を見せています。マイコン LSI を仮に上位機種 (16, 32 ビット)、中位機種 (8 ビット)、下位機種 (シングル・チップ) に分けて流れを見ると、次のような動向が見られます。すなわち、上位機種では高性能化の追求が行われ、32 ビット機種が登場してきており、中位機種では周辺機能を取り込んだシステム集積化による性能・コスト向上の追求が、また下位機種ではシステム集積化とともに各応用分野を指向した専用化が行われています。また、すべてに共通した事項として、低電力化を指向したプロセスの CMOS 化が顕著な動向として挙げられます。上記のハードウェアの流れに加え、マイコンの世界で無視し得ないものとして、いわゆる標準 OS とそのうえで走るソフトウェア財産の流れがあります。1978 年ころ、8 ビット・マイコン用として登場したオペレーティングシステム CP/M はそのうえに数千点の応用ソフトウェアが蓄積され、標準 OS としての位置を占めました。その後、そのソフトウェア財産は CP/M-86、MS-DOS を通じ、16 ビット・マイコンにも継承されています。CP/M の走る 8 ビット・マイクロプロセッサで中心的な地位を占めているのが Zilog 社の Z80 です。現在でも多くの 8 ビット・パソコンが Z80 を使用しているのは皆様御存じのとおりです。64180 は Z80 を、CP/M がより効率的に走ることも含め機能強化したシステム集積型の 8 ビット・マイクロプロセッサです。

64180 の開発には以下に述べるようないきさつがあります。1982 年の暮れ、米国 Microfuture 社のトム・カントレル (推薦のことば筆者、当時、日立アメリカ社) と筆者は、当時、CP/M-68K の共同開発を行っていた米国ディジタルリサーチ社のソフトウェア技術者達と仕事の後に、雑談をしていました。その際、話題となったのが、8 ビット・プロセッサのアドレス空間の制限一俗にいう 64K バイトの壁、さらに当時同社で開発中であった CP/M-Plus (V3.0) におけるバンク切替方式による本問題への対策、あるいは OS 効率化のための RAM ディスク

は し が き

などでした。その際、一同の間で期せずしてひらめいたのが、Z80 を機能強化し上記問題を解決し、CP/M-Plus を効率的に走らせるプロセッサを開発してはという発想でした。本件はその後日立の技術者達に取り上げられ、具体化のための検討が行われ、さらに広いユーザの意見を聞いた結果できあがったのが 64180 のスペックなのです。その後の検討により数々の機能が付け加えられていますが、64180 の MMU の方式、DMA によるメモリ↔メモリのブロック転送などには、上記の議論の結果を色濃く残しています。64180 は、ソフトウェア技術者の提案を端緒としてこの世に出たプロセッサなのです。

この 64180 プロセッサですが、詳細は以下の章に譲るとして、大きくは次のような特徴をもっています。MMU、DMA を内蔵し、8ビット・プロセッサで問題のアドレス空間は 1 MB まで拡大されています。ネットワーク時代への備えとして、シリアル・インタフェース 3 チャンネルをもっています。命令セットは Z80 に対し、アップワードにコンパチブルで新たに乗算を含む 12 種の命令が付加されています。スリープ命令は CMOS プロセスとあいまって、低電力化の効果を一段と発揮するものです。何よりも、Z80 コンパチブルの命令セットとバンク切替方式の MMU は、CP/M-Plus (V3.0) を効率的に走らせ、そのうえのソフトウェア財産を活用できる点で、64180 の最大の特徴といえましょう。以上の点よりもわかるように、64180 は 8 ビット機ではありますが色々な点で機能強化され、16 ビットへの橋渡しをするマイクロプロセッサなのです。

本書は、主として Z80 を使用したことのある方々を対象としています。その意味で Z80 と相違する点、付加した機能について特に注意して説明するよう心掛けました。前半に主としてハードウェア機能を、後半には CP/M-Plus (V3.0) とのインタフェースを含む応用例について述べています。本書が皆様の 64180 の応用を進める際に良い手ほどきとなることを期待します。

昭和 62 年 12 月

編 者 し る す

目 次

— 第 I 編 C P U —

1. 64180 の 概 要	
1・1 システムオンチップ技術の流れ	4
1・2 64180 の特長と内蔵機能	6
1・3 ソフトウェアの継承性	9
1・4 Z80 との相違	11
1・5 応 用 分 野	13
1・6 開 発 環 境	15
1・7 ファミリー展開	18
2. CPU アーキテクチャ	
2・1 レジスタ構成	22
2・2 フラグレジスタ	24
2・3 命令の構成	26
2・4 データの配置	29
2・5 アドレッシングモード	32
2・6 メモリ空間と I/O 空間	34
2・7 命令の分類	36
3. CPU タイミング	
3・1 メモリリードサイクル・ライトサイクル	40
3・2 I/O リードサイクル・ライトサイクル	42
3・3 オペコードフェッチサイクルと命令の実行	45
4. メモリマネジメントユニット	
4・1 MMU の概念	48

4・2	MMU の動作	50
4・3	MMU の使用例	55

— 第II編 周辺機能 —

5.	タイマ	
5・1	タイマの構造	64
5・2	リロードタイマの動作原理	66
5・3	タイマの使用例	68
6.	非同期方式シリアル I/O	
6・1	シリアルコミュニケーションインタフェースの分類	72
6・2	ASCI の構成と機能	74
6・3	ASCI のデータフォーマットおよび転送速度	76
6・4	ASCI のエラーチェック機能	78
6・5	モデム制御信号	80
6・6	ASCI の使用例	82
7.	クロック同期方式シリアル I/O	
7・1	CSI/O の構成と機能	86
7・2	CSI/O の動作原理	88
7・3	CSI/O の使用例	90
8.	WAIT リフレッシュコントローラ	
8・1	ウェイトの動作原理と使用法	94
8・2	リフレッシュコントローラの動作原理	96
8・3	リフレッシュタイミングと DRAM とのインタフェース	98
9.	割込み	
9・1	割込みとは	102
9・2	ベクタ方式	104
9・3	マスカブルな割込み	107

9・4	$\overline{\text{INT0}}$ モード 0	109
9・5	$\overline{\text{INT0}}$ モード 1	112
9・6	$\overline{\text{INT0}}$ モード 2	113
9・7	ノンマスカブルな割込み	115
9・8	RET 命令, RETI 命令, RETN 命令	117
10. 特殊動作モード 		
10・1	リセット	120
10・2	ホルトモード	122
10・3	バスリリースモード	124
10・4	低消費電力モード	126
11. DMA コントローラ 		
11・1	DMAC とは	130
11・2	DMAC の構造	133
11・3	DMAC の転送モード	135
11・4	チャネルの優先順位	140
11・5	DMAC とブロック転送命令	143
11・6	DMAC の使用例	145
 — 第III編 応 用 — 		
12. メモリインタフェース 		
12・1	SRAM の接続	150
12・2	擬似 SRAM の接続	151
12・3	DRAM の接続	152
12・4	EPROM の接続	154
12・5	$\overline{\text{WAIT}}$ 信号発生回路	155
13. I/O インタフェース 		
13・1	80 系周辺 LSI との接続	158
13・2	Z80 周辺 LSI の接続	160

13・3	68 系周辺 LSI との接続	163
13・4	FDC の接続	165
13・5	CRTC の接続	167
13・6	LCTC の接続	169
14. 新規命令		
14・1	新規命令の概要	172
14・2	入出力命令の使用例	174
14・3	論理演算命令 (TSTIO 命令) の使用例	176
14・4	MLT 命令の使用例	178
15. 180 カードパソコン		
15・1	カードパソコンについて	182
15・2	システム構成	184
15・3	I/O メモリインタフェース	186
15・4	CP/M-Plus	188
15・5	BIOS の構成	190
15・6	MMU と CP/M-Plus のバンク	192
15・7	DMA 転送とバンク	195
15・8	CP/M-Plus 起動時のメモリマップの変化	197
15・9	64180 内蔵 I/O と BIOS	199
付 録		
1	命令一覧	201
2	内蔵 I/O レジスタ一覧	219
3	端子名一覧	229
4	R1 版と Z 版との違い	230
5	180 カードパソコンのハードウェア仕様	232
参考文献		234
索引		235

第 I 編

C P U

64180 は、Z80 と上位コンパチブルな命令セットをもつ CPU を中心に、DMA コントローラや MMU などの周辺機能をワンチップに集積化したマイコン LSI です。第 I 編では、追加命令を中心とした CPU アーキテクチャおよびメモリアンタフェースタイミングについて説明します。また、8ビットマイコンの弱点である、メモリ空間の制約（64KB 以下）を解決するために導入されたバンク方式 MMU の詳細について最後に説明します。いずれの項目も、64180 を使ったシステムのソフト/ハード設計には必須の基本事項です。

1. 64180の概要

64180 は、DMA コントローラや MMU などのシステムの構成に必要な周辺機能を内蔵した、8ビットでは世界で初めての「システム集積型」マイコンです。CPU は、Z80 と上位互換性のある命令セットをもっていますが、ソフト/ハードの両面で Z80 から改良されており、16ビット CPU にせまる性能を実現しています。各種の開発ツールもそろい、一方、製品のファミリー化も計画されており、8ビットの代表的 CPU のひとつとして息の長い製品となるものと思われます。

1・1 システムオンチップ 技術の流れ

〔1〕 **マイコン CPU の発展** 世界初のマイコン LSI4004 から10数年、この間半導体技術の進歩にささえられて、マイコン LSIは目覚ましい発展を遂げ、今や 32 ビットマイコンが実用化されるに至っています。32 ビットマイコンの華々しい開発競争に、ともすれば世間の注目が集まりがちですが、実際にはマイコンの出荷数量の 90 %以上を 8 ビット以下のマイコンが占めており、この分野でも目覚ましい進歩が続いています。32 ビットマイコンを実現する半導体技術を、性能追求ではない別の目的に使ったらどうなるのか？それがシステムオンチップ化です。

〔2〕 **システムオンチップ化** 最初にシステムオンチップを実現したのは、4 ビットのシングルチップマイコンです。主に、ROM、RAM を内蔵してワンチップでコンピュータの機能を実現しました。シングルチップマイコンは、家電製品などに内蔵されてメカの制御などに使われています。たとえば、VTR などでは、マイコンの存在がなければ製品が成立しえないほど重要な部品となっています。一方、これらのシングルチップマイコンの流れに対し、1980 年代に入って少し異なった形のシステムオンチップ化を実現したマイコンが現れました。それが 80186 です。このマイコンでは、メモリ以外でシステムの構成に必須の機能を集積化しています。いわば「システム集積型」マイコンといえます。

〔3〕 **システム集積型マイコン** なぜこのような「システム集積型」マイコンが出現したかは、マイコンの応用分野を考えてみるとわかります。表 1・1 に示すように、マイコンの応用は大きく分けて、メカ制御中心の**コントローラ型**とデータ処理中心の**データ処理型**に二分できます。コントローラ型は、必要なメモリサイズが比較的小さくワンチップに集積化が可能です。すなわち、シングルチップマイコンとなるわけです。一方、データ処理型の応用の場合には、扱うデータ量が 64 KB 以上のことが多く、現在の技術ではワンチップ化が困難です。そこで、メモリ以外でシステム構成によく使われる、DMAC、タイマ、シリアルなどの標準的周辺機能を内蔵したマイコンが出現したわけです（図 1・1 参照）。64180 は、8 ビットでは世界初のシステム集積型マイコンです。

1・1 システムオンチップ技術の流れ

表 1・1 マイコンの応用分野の分類と使用されるマイコン

項 目 \ 分 類	データ処理型	コントローラ型
特 長	データ処理中心 コンピュータ・ライク	メカ制御中心 機器組込み型
応用機器の例	パソコン、ワークステーション タイプライタ、ワープロ	家電品(VTR, エアコン など) 自動車エンジン制御 ロボット
必要なメモリ空間	大きい (64 KB 以上)	小さい (64 KB 以下)
内蔵機能への要求	メモリ以外のシステム構成に 必要な機能の内蔵	all-in-one
必要な内蔵機能	タイマ、シリアル、DMAC MMU、キャッシュメモリ リフレッシュコントローラ ウェイトコントローラなど	ROM, RAM タイマ、シリアル A/D 変換器 I/Oポートなど
使われるマイコンの流れ	マルチチップマイコン ↓ システム集積型マイコン	シングルチップマイコン ↓ ASIC マイコン

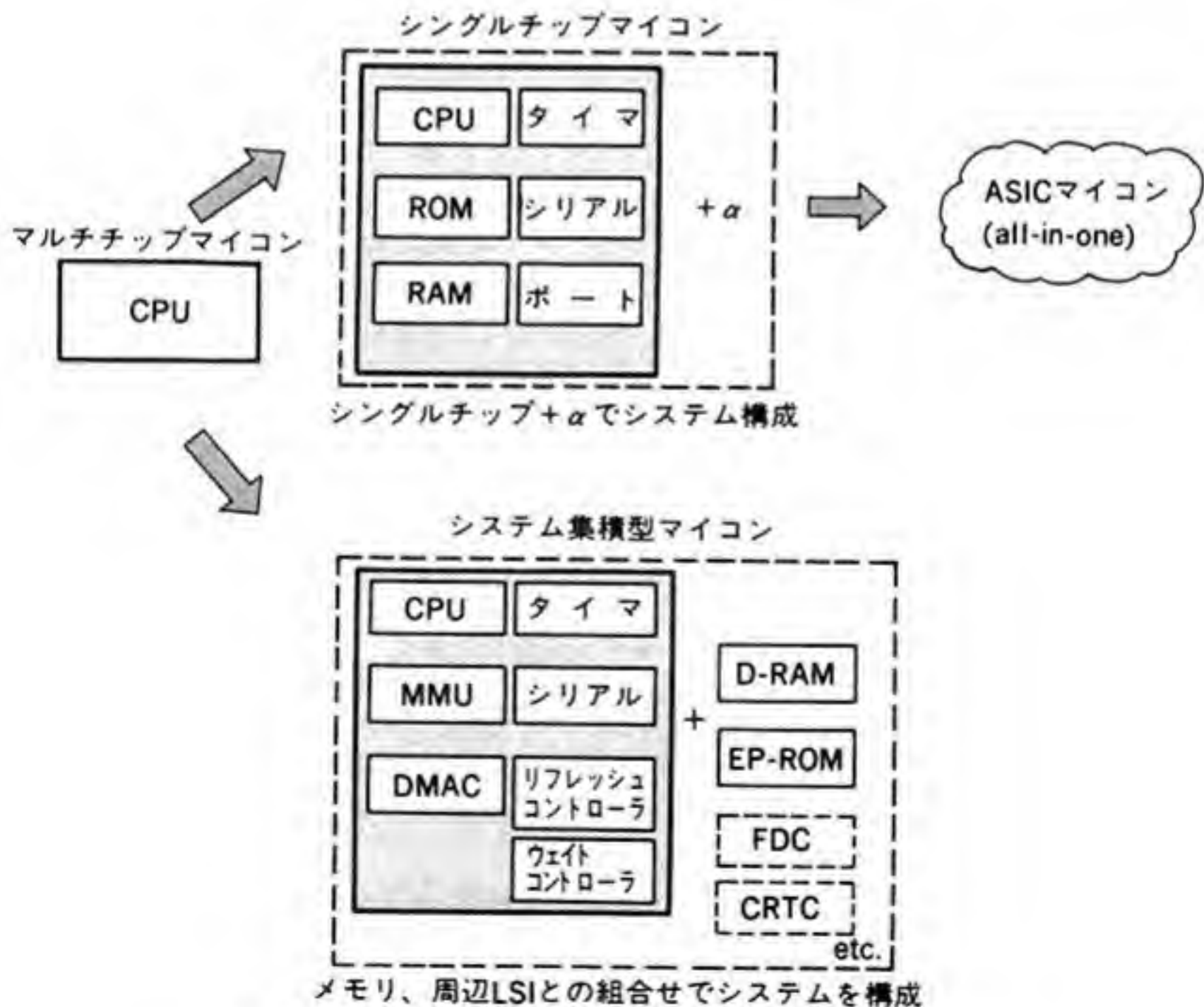


図 1・1 シングルチップマイコンとシステム集積型マイコン

1・2 64180の特長と内蔵機能

〔1〕 なぜ8ビットシステム集積型マイコンか？ データ処理型の応用では16ビットマイコンが主流になるのではないかと考えられる方も多いと思いますが、必ずしもそうはならない理由があります。まず LSIコストの問題があります。また、16ビット幅のバスはプリント基板上で場所をとりコスト高となります。まだまだメモリにおいて×16ビット出力のものは数少ないという問題もあります。また、周辺 LSIは大部分が8ビットバスです。メモリ空間サイズを除けば、性能的には8ビットで十分という応用も数多くあります。また、8ビットでの多くのソフトウェア財産の存在も見逃せません。

表 1・2 主要性能と機能

項 目		仕 様	
プ ロ セ ス		CMOS	
特 性	消 費 電 力 (typ) $(V_{cc}=5V)$ $(f=4MHz)$	通 常 動 作 時	50mW
		ス リ ー プ・モ ー ド 時	30mW
		シ ス テ ム・ス ト ッ プ 時	12.5mW
	最 小 命 令 実 行 時 間	300ns ($f=10MHz$)	
機 能	命 令 体 系	Z80上位互換	
	割 込 み	外部4本、内部8本	
	ア ド レ ス 空 間	1Mバイト (512バイト…DIPパッケージの場合)	
	DMA コ ント ロ ー ラ	2チャンネル ○メモリ↔メモリ ○メモリ↔I/O (メモリ・マップI/O) 最大転送レート 1.3 Mバイト/秒 ($f=8MHz$)	
	非 同 期 シ リ ア ル 通 信 イ ン タ フ ェ ー ス (ASCII)	2チャンネル ○8種のデータ・フォーマット ○3種のエラー・フラグ ○モデム制御信号あり	
	ク ロ ッ ク 同 期 式 シ リ ア ル I/O ポ ー ト (CSI/O)	クロック同期シフト・レジスタ1チャンネル 最大ボーレート 300Kボー ($f=6MHz$)	
	タ イ マ	16ビット・リロード・タイマ2チャンネル タイマ出力あり("0", "1", トグル出力)	
	バスウェイトコントローラ	外部端子による制御 内部レジスタによる制御 (メモリ, I/O) 独立に設定可	
	リフレッシュコントローラ	プログラマブル非同期リフレッシュ 8ビットのリフレッシュ・アドレス (256K DRAM 対応)	
	バスインタフェース	80系/68系(63系)とも可能	

〔2〕 16ビットにせまる8ビットマイコン 64180は、命令体系に8ビットの世界標準ともいえるZ80と上位互換性のある命令体系が採用されており、既存のソフトウェア財産を活用できます。また、8ビットマイコンの弱点であったアドレス空間の制限(64Kバイト以下)は、内蔵のバンク方式のMMUによって命令の互換性を保ちながら最大1Mバイトまで拡大されています。

乗算命令の追加、10MHzの高速動作とあいまって、まさに16ビットの性能にせまる8ビットマイコンといえます。

〔3〕 内蔵機能 CPU、MMU以外の内蔵機能としては、DMAコントローラ、シリアルインタフェース、タイマ、リフレッシュコントローラなどがあります(図1・2、表1・2参照)。特に、シリアルインタフェースが3チャンネル内蔵されているところに大きな特長があります。これは、今後の応用機器を考えると分散処理の方向へ動くと考えられ、その場合各機能ごとにマイコンが入るこ

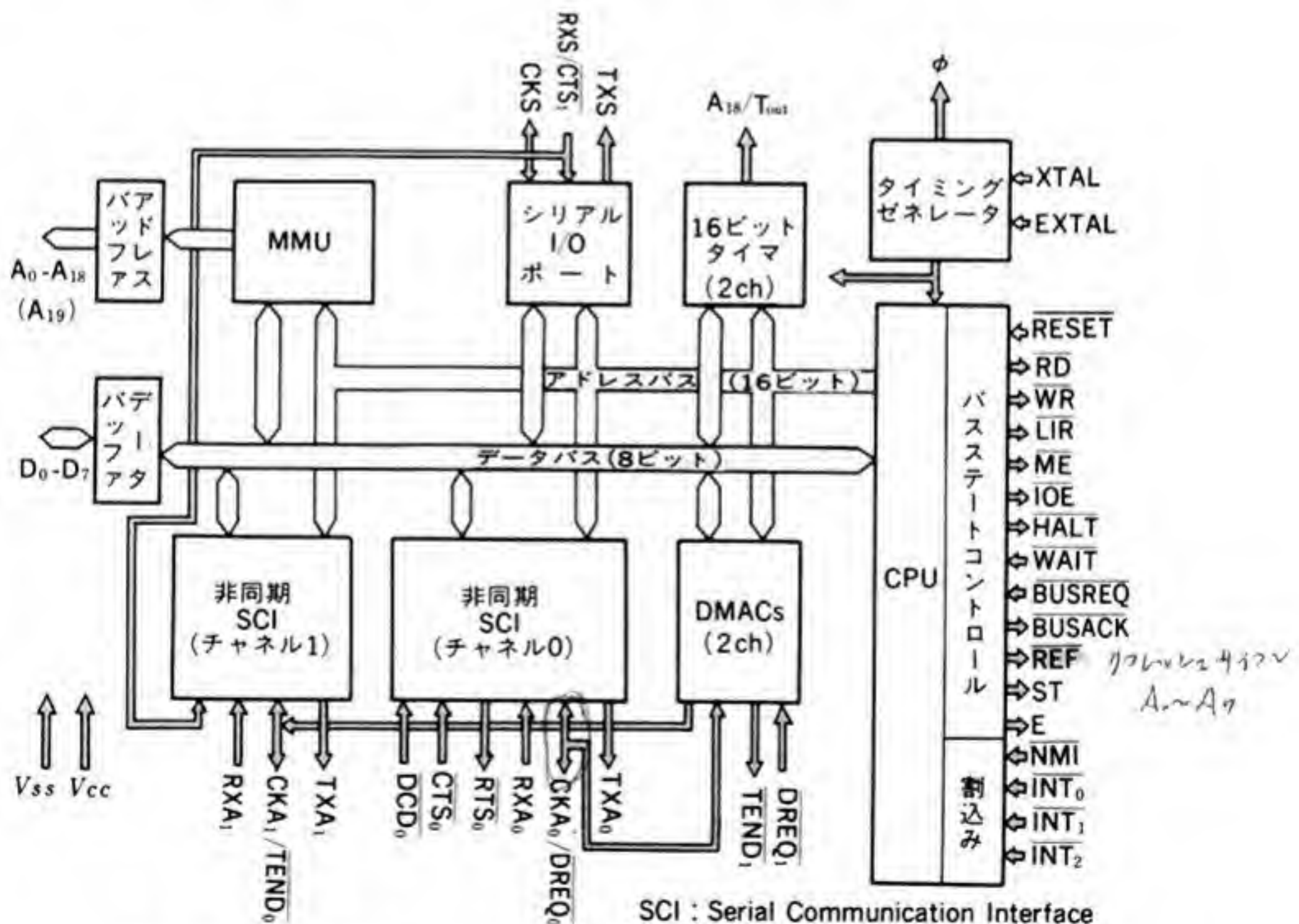


図1・2 ブロック図(内部アドレスは16ビット、MMUにより19ビット(20ビット)に拡張された物理アドレスが出力される)

1. 64180 の 概 要

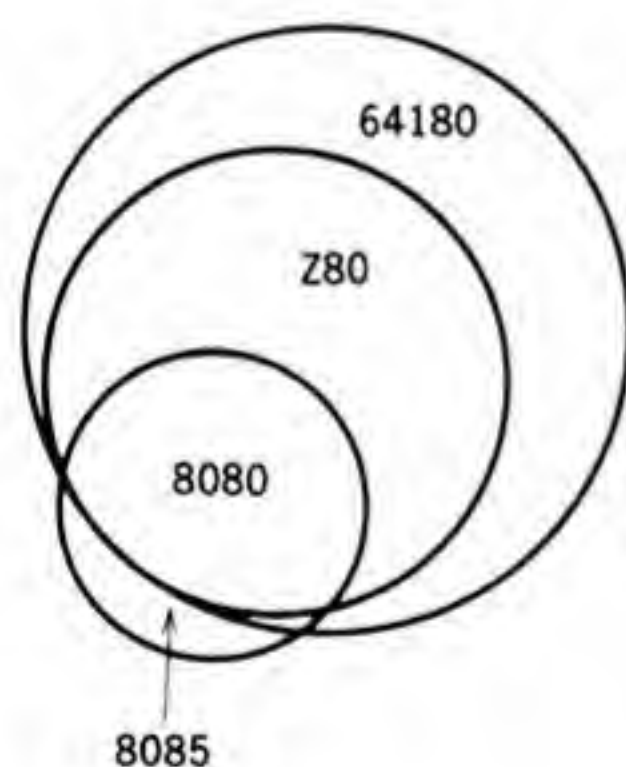
とが予想されます。その際、各マイコン間のインタフェースはシリアルインタフェース経由で行われるのが普通です。上位 CPU との接続、サブ CPU との接続などを考えると、シリアルインタフェースは複数必要となります。

リフレッシュコントローラ、ウェイトコントローラの内蔵も大きな特長です。これらが内蔵されていることにより、ダイナミックメモリや擬似スタティックメモリとのインタフェースはごくわずかの外付 IC で済み、驚くほど簡単に実現できます。

1・3 ソフトウェアの継承性

〔1〕 80系マイコン 8ビットのマルチチップマイコンで最も広く使われているものが Z80 CPU です。国内では8ビットパソコンの大部分や MSX パソコンのメイン CPU に Z80 が使われているのは承知の事実です。この Z80 の命令セットは 8080 の命令を包含し、オブジェクトレベルで互換性を保ちながら機能拡張されています。64180 ではさらに、Z80 に上位互換性を保持しながら7種類12個の命令が追加されています。

〔2〕 64180 の追加命令 64180 で強化された命令のうち、なんといっても効果が大きいものが乗算命令 (MLT 命令) です。これにより8ビットの乗算は約1桁速くなりました。また、スリープ命令 (SLP 命令) は CMOS の特長を生かした命令です。64180 は CMOS で作られているので、NMOS の標準的な Z80 に比べて消費電力はもともと約 1/10 ($f = 4 \text{ MHz}$ 時、50mW) ですが、この SLP 命令を使えば CPU はスタンバイ状態となり、さらに通常動作時の約 1/4 ($f =$



ニモニック	動 作	備 考
SLP	Sleep	
MLT ww	$ww_R \leftarrow ww_{Hr} \times ww_{Lr}$	ww = BC, DE, HL
INO g, (m)	$g \leftarrow (m)$	I/O アドレスの $A_{15} \sim A_8$ は 00H
OUTO (m), g	$(m) \leftarrow g$	I/O アドレスの $A_{15} \sim A_8$ は 00H
OTIM	$(C) \leftarrow (HL), HL \leftarrow HL + 1$ $C \leftarrow C + 1, B \leftarrow B - 1$	I/O アドレスの $A_{15} \sim A_8$ は 00H
OTIMR	$(C) \leftarrow (HL), HL \leftarrow HL + 1$ $C \leftarrow C + 1, B \leftarrow B - 1$ Repeat until B=0	I/O アドレスの $A_{15} \sim A_8$ は 00H
OTDM	$(C) \leftarrow (HL), HL \leftarrow HL - 1$ $C \leftarrow C - 1, B \leftarrow B - 1$	I/O アドレスの $A_{15} \sim A_8$ は 00H
OTDMR	$(C) \leftarrow (HL), HL \leftarrow HL - 1$ $C \leftarrow C - 1, B \leftarrow B - 1$ Repeat until B=0	I/O アドレスの $A_{15} \sim A_8$ は 00H
TSTIO m	$(C) \wedge m$	I/O アドレスの $A_{15} \sim A_8$ は 00H
TST g	$A \wedge g$	g はレジスタ
TST m	$A \wedge m$	m は 8 ビットデータ
TST(HL)	$A \wedge (HL)$	

図 1・3 80系マイコンの命令セットの関係と 64180 の追加命令一覧表

1. 64180 の 概 要

4 MHz 時、12.5 mW) まで下げることができます。スリープ命令は CPU を間欠動作させて平均消費電流を抑えたいような応用に便利です (図 1・3)。

〔3〕 CP/M80 OS 8 ビットマイコンの代表的 OS に、米国デジタルリサーチ社の CP/M80 があります。CP/M は 8080 系の 8 ビット CPU を使うパソコンのほとんどの機種に採用されており、世界的標準となっています。パソコン以外でもワープロ、タイプライタなど CP/M で動いているものが数多くあります (図 1・4)。

CP/M には、一般的な CP/M2.2 [Ver 2.2] と、機能改良された CP/M PLUS [Ver 3.0] があります。特に CP/M PLUS では、バンクモードというメモリ空間の拡張機能がサポートされています。このバンクモードは、64180 のバンク方式 MMU を使えば外付回路なしで実現することができます。64180 は CP/M 2.2 のみならず、CP/M PLUS の搭載にも最適の CPU といえます。

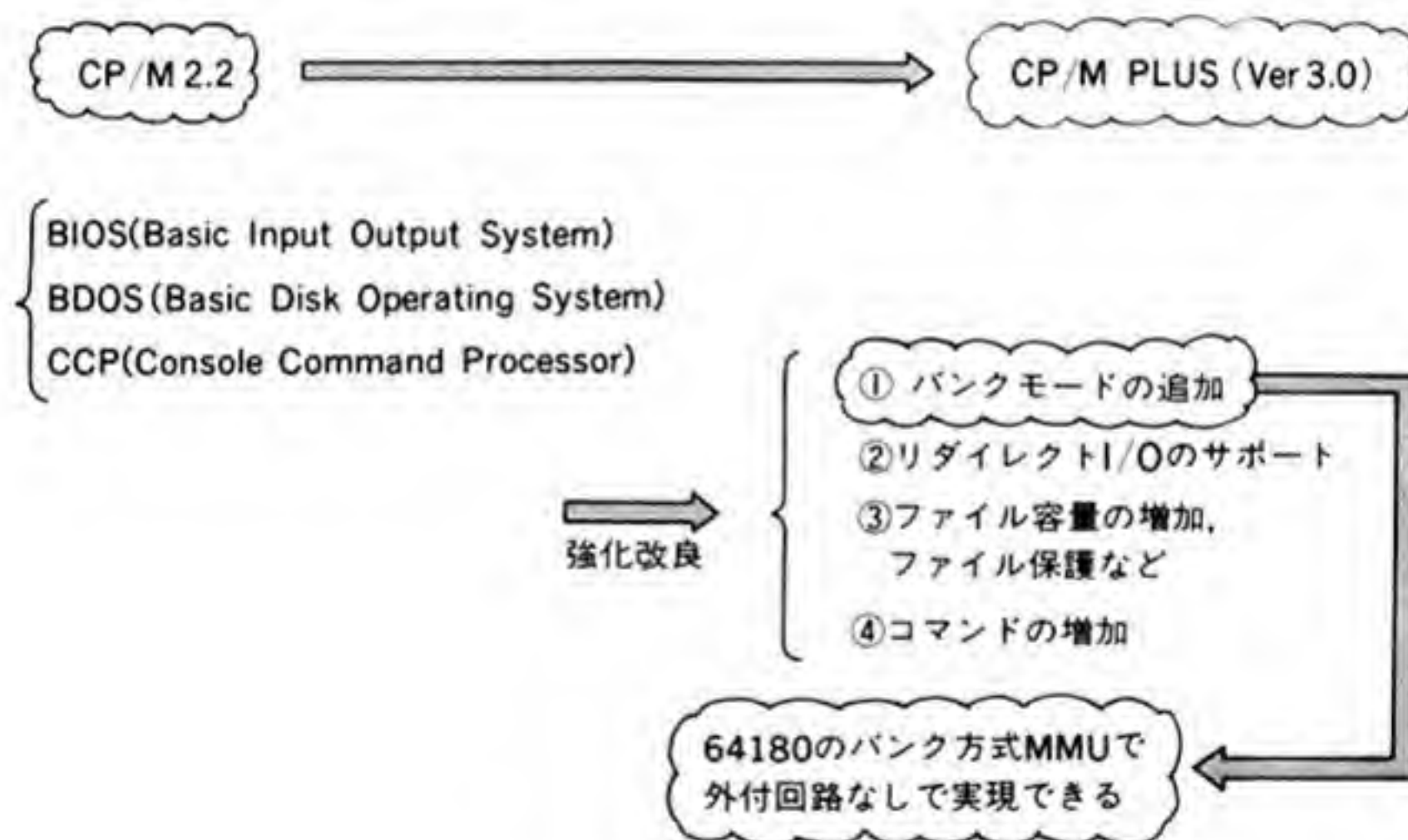


図 1・4 CP/M80 OS の発展

1・4 Z80 と の 相 違

64180 には前節で述べたように、Z80 に 12 個の命令が追加されています。また、各命令の実行サイクル数も約 10～20 % 短縮されており、6 MHz バージョンで Z80 の 7～8 MHz 版に匹敵する処理能力をもっています。

端子は、Z80 に相当する端子が備わっていますが名前は一部異なっています。

$$\overline{M}_1 \longleftrightarrow \overline{LIR}, \overline{MREQ} \longleftrightarrow \overline{ME}, \overline{IOREQ} \longleftrightarrow \overline{IOE}$$

バスタイミングは基本的には Z80 と同じですが、細部は異なります。特に、次に述べるリフレッシュ方式の違いにより、命令フェッチサイクルが 3 ステートとなっている点が一番大きな違いです。Z80 ではこれが 4 ステートとなっており、第 4 ステートでリフレッシュを行っています。その他にも \overline{ME} 、 \overline{IOE} 、 \overline{RD} のタイミングについて細かい違いがあります（付録参照）。

リフレッシュサイクルの挿入方法が改善されて、ダイナミック RAM とのインタフェースが格段にやりやすくなったことは、64180 の大きな利点のひとつです。Z80 の場合、 \overline{M}_1 サイクルで必ずリフレッシュサイクルが入るため、特に高速版ではタイミング設計が非常に難しく、また、リフレッシュサイクルが不必要に頻繁に入ってしまうという問題がありました。64180 では、リフレッシュサイクルは命令実行とは非同期に、一定サイクル間隔ごとに入る方式に改められています。リフレッシュ挿入の間隔や、リフレッシュサイクルの長さなどがプログラム指定でき、周波数が変わった場合も最適のリフレッシュ間隔を選ぶことができます。

割込み方式の改良も大きな利点のひとつです。Z80 の場合、Z80 系の周辺 LSI を使う場合はデイジーチェーン方式が使えて簡単にシステムを組めますが、Z80 系以外の周辺 LSI の場合は設計が難しくなります。64180 では、Z80 と同一の割込み端子 (\overline{INT}_0 、 \overline{NMI}) に加えて割込み端子が 2 本追加されており (\overline{INT}_1 、 \overline{INT}_2)、これらに対応するベクタは内部で自動的に発生する方式をとっております。このため、Z80 系以外の周辺 LSI からの割込みの接続が非常に簡単にできます（表 1・3）。

1. 64180 の 概 要

表 1・3 Z80 との相違点

	Z80	64180
追 加 命 令	—	6種類12命令追加 (SLP,MLT,TST,INO,OUTO,OTIM)
命令実行サイクル数	—	約10～20%サイクル数短縮
端 子 名	M ₁ , MREQ, IOREQ	LIR, ME, IOE
クロック発振器	外部クロック入力	発振器内蔵（水晶またはセラミックの振動子をつけるだけでよい）
バスタイミング	—	ME, IOE, RD の出力タイミングが異なる
リフレッシュサイクルの挿入方法	各命令フェッチサイクルの第4サイクルに必ず挿入（命令同期式）	一定サイクル間隔ごとに命令実行に非同期に挿入（命令非同期式）リフレッシュインターバルを周波数によらず一定にできるので不必要に頻繁なリフレッシュを避けることが可能。リフレッシュサイクルの長さも2または3ステートが選べ、リフレッシュサイクルを全く挿入しないモードも可能
割 込 み	2本： $\overline{\text{NMI}}$, $\overline{\text{INT}}$ デジチェーン方式	4本： NMI , INT_0 , INT_1 , INT_2 デジチェーン方式も可能（ INT_0 ）だが、 INT_1 , INT_2 および内蔵I/Oからの割込みはベクタ番号を内部で発生。固定ベクタ、固定優先順位の割込みコントローラを内蔵
アドレス空間	64KB	512KB（面実装パッケージでは1MB） 内蔵のバンク方式MMUによる論理空間（64KB）から物理空間への写像により最大1MBを直接アクセス可能
低消費電力モード	—	スリープモード/システムストップモード
ウェイトステート制御	$\overline{\text{WAIT}}$ 端子のみ	$\overline{\text{WAIT}}$ 端子の他にプログラマブルウェイト可能
内 蔵 I/O 機 能	—	DMAC, タイマ, シリアル

代表例

命 令	Z80	64180
ADD A, (HL)	7	6
ADD IX, xx	15	10
BIT b, (HL)	12	9
LD A, (BC)	7	6
LD (BC), A	7	7
LD ww, nn	20	9
JR _i	12	8

1・5 応 用 分 野

以上述べてきたような優れた特長をもつ 64180 の応用としては、次のような分野が考えられます（表 1・4）。

(1) 従来から 80 系のマイコンを使用しており各種のソフトウェア財産が存在するが、アドレス空間が 64 K バイトでは不足してきており、**16 ビットを使うべきかどうか迷っている分野**。ハンディー日本語ワープロ、プログラマブルシーケンサ、FAX といった機器がこれにあたります。特に、ダイナミック RAM や擬似スタティック RAM を使うシステムや、All-CMOS 化して低消費電力動作を狙う分野に適しています。

(2) 従来から 80 系のマイコンを使用しており、アドレスは 64 K バイト以下でよいが、タイマ、シリアルなどの周辺 LSI を多用している分野、**機器のサイズの関係で基板をできる限り小さくしたいし、消費電力ももちろん下げたい**。このような分野では、64180 の内蔵機能が基板サイズ縮小に大きな効果があります。

(3) 従来 8 ビットシングルチップマイコンを使っていたが、制御が複雑となり、**メモリ容量が増加しシングルチップに入りきらなくなっている分野**。しかし、できる限りシングルチップマイコンと同じように簡単な回路構成としたい。たとえば、電子タイプライタ、電子はかり、券売機、PPC などがこれにあたります。

(4) 応用機器の性質として、**シリアルインタフェースが少なくとも 2 本以上必要な応用**。たとえば、端末、POS、モデム制御、自動車電話、キーテレホンなどがこれにあたります。

表 1・4 応用分野と応用機器の例

分 野	応 用 機 器
民 生	電子楽器、無線機
産 業	プログラマブルシーケンサ、電子はかり、自動販売機、券売機、自動車パネル表示器
OA、情報	MSX パソコン、プリンタ、電子タイプライタ、ハンディー日本語ワープロ、端末、POS、ECR、XY プロッタ、PPC
通 信	FAX、キーテレホン、PBX、モデム制御、パソコン通信、自動車電話

1. 64180 の 概 要

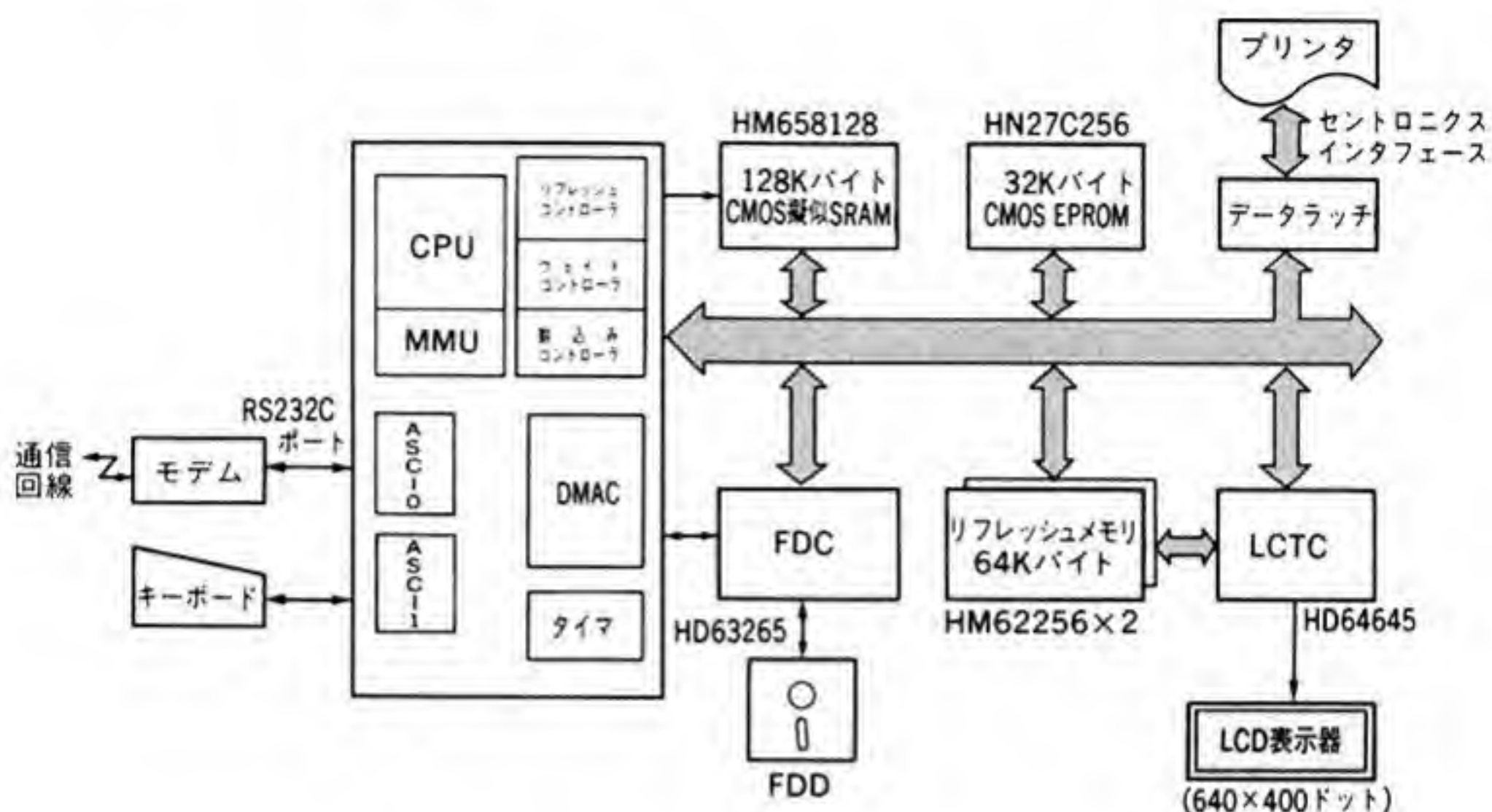


図 1・5 液晶表示つきワープロ/パソコンへの応用例

以上のような例の他にも、一般に高速処理、大きなアドレス空間、タイマ、シリアルなどの周辺機能が必要な応用など、幅広い応用が考えられます。Z80と同様に8ビットマイコンの標準品として、息の長い製品となるものと考えられます(図1・5)。

1・6 開 発 環 境

どんなに優れたマイコン LSI であっても、開発装置がなければシステムを開発することはできません。その点 64180 は、半導体メーカーである日立製作所以外にも、数多くの開発ツールメーカーからサポートツールが発売されています。廉価普及版から高性能高機能型まで用途に応じて選ぶことができます(表 1・5)。

開発ツールの形態は大きく分けて、専用の開発装置をホストとするタイプと汎用のパソコンやミニコンをホストとするタイプに分けられます。専用型の代表例は YHP 社の 64000 などがそれです。専用型はホストが開発装置専用なのでコスト的に割高となりますが、一般に、機能、性能、使い勝手に優れています。

一方、汎用ホスト型は日立製作所の ASE などがそれにあたります。汎用ホスト上でエディット、クロスアセンブルを行い、RS232C のシリアルインタフェース経由でロードモジュールを専用のリアルタイムエミュレータ上にロードしデバッグを行います。汎用型はコスト的には安くなりますが、大きなプログラム開発になると RS232C によるダウンロード時間がネックとなる場合があります。

最近では汎用型でも、ホストをたとえば PC9801 に固定しパソコン内に専用基板を差し込んで、エミュレータとのインタフェースを平行バスで行い、ダウンロード時間のネックを解消した製品もあり(岩崎技研工業社製の PROICE など)ですが、平行バス方式は、ホストが限定されており専用型と汎用型の中間といえます。

また、各社ともユーザの負担をできる限り下げするために、システムの一部のボードなどを取り替えることによって各種のマイコンに対応できるようにしてあるものが増えてきています。

予算の関係で 64180 用の開発ツールを購入できない場合には、CP/M ののっているパソコンと Z80 用のアセンブラでも十分開発は行えます。たとえば、Microsoft 社の MACRO80 のようにマクロ機能つきのアセンブラであれば、64180 の拡張命令をマクロ定義して使うこともできます。また、CP/M に内蔵のデバッグでも十分プログラムデバッグはできます。不足分はロジックアナライザで補えばよいでしょう(図 1・6)。

1. 64180 の 概 要

表 1・5 64180 開発ツール一覧表

メーカー名	ホストコンピュータ	リアルタイム エミュレータ	クロスソフトウェア		
			アセンブラ	シミュレータ	C コンパイラ
日立製作所	VAX-11 SD200 IBM-PC	○	○	—	○
横河ビューレット パソコン (YHP)	HP64000	○	○	—	—
岩崎技研工業	PC9801 IBM-PC	○	○	—	○
ソフィアシステムズ	SA3000 ES1000	○ (SA600)	○	—	—
横河電機 (YEW)	HP64000 VAX-11, MDS パソコン	○ (3503)	※	※	※
ZAX コーポレーション	THE BOX-ER PC9801, IBM-PC	○	○	—	○
ユニダックス	PC9801	○	※	※	※
日本データ ゼネラル	ECLIPSE MV ファミリー	(○) 他社製品接続	○	○	○
アドテックシステム サイエンス	—	○	—	—	—
MICROTEC (日本マイクロ) (テックリサーチ)	VAX-11 PC9801 IBM-PC, 他	—	○	○	○
コアデジタル	PC9801 IBM-PC, IBM5550 B16, HP64000 VAX-11	○	※	※	※
アンリツ	VAX-11, PC9801 IBM-PC HP64000 CP/M	○	※	※	※
ソニー・ テクトロニクス	ロジックアナライザ 1240/1241 型 +プローブ	—	—	—	—

※ マイクロテック・リサーチ社製他、市販ソフトを使用

(注) 上表の他に、下記各社製の 64180 用ソフトウェアが販売されています(() 内：販売会社)

エイ・ディ・シー(ケミコン・セミコンダクターズ・セールス)、AVOCET(工人舎)、ソフトウェアアシスト(住友商事)、ソフトマート、BSO(エーエスアールインターナショナル)、2500AD (アドテックシステムサイエンス)、IAR(ライフ・ボード)、ファームウェアシステム

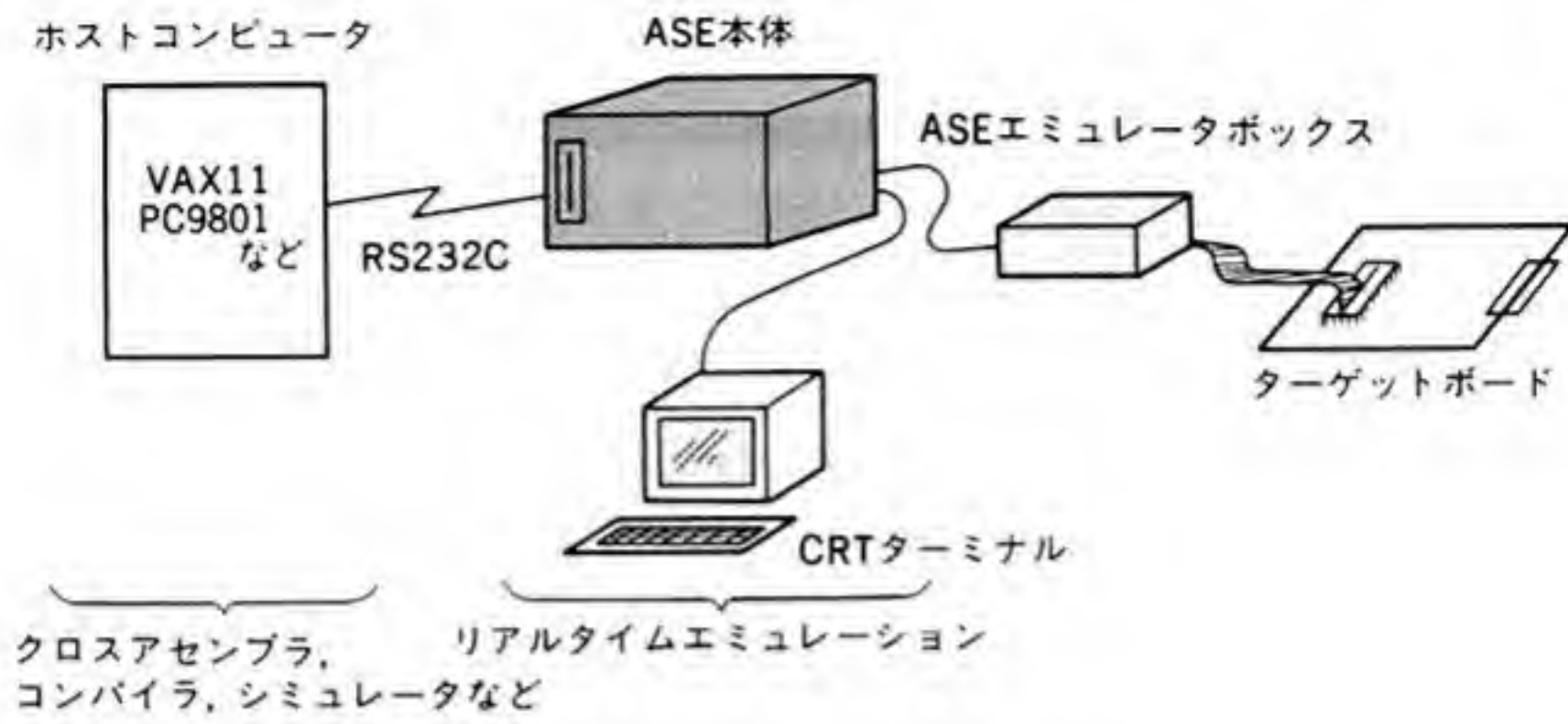


図 1・6 64180 開発環境の例

1・7 ファミリー展開

64180が発表されて約3年がたちましたが、この間、スピードアップ(10MHz)、メモリ空間のアップ(1Mバイト)をはかり、初期のバージョンの不具合点の改良を行った“R1”バージョン、また、Z80系の周辺LSIとのインタフェースを改良した“Z”バージョンなど改良の努力が続けられています。64180は、Z80の本家のZilog社からもセカンドソース供給されています。“Z”バージョンはZilog社では“Z180”と呼ばれています。

1987年春には注目すべき新製品が発表されました。64180の全機能はそのまま16KバイトのEPROM、512バイトのRAM、タイマ、コンパレータなどをオンチップ化した647180Xがそれです。647180Xは日立製作所が提唱したZTAT(Zero Turn Around Time)マイコンシリーズのひとつであり、いわゆるOTP(One Time Programable)マイコンです。ZTATマイコンは、紫外線消去のEPROM内蔵マイコンをプラスチックパッケージに封止したものです。1回しかプログラムできないかわりに、従来のマスクROM型のシングルチップマイコンに近い価格で手に入れることができます。ZTATマイコンは、少量多品種生産やプログラムバグが収束するまでの量産立上り時期に使うと効果的です。647180Xは、64180ファミリーの応用範囲を大きく広げる製品といえます。

64180系としては、このほかにもZTATシリーズの品種追加、通信機能の強化版などの標準品展開、およびASIC展開(カスタム対応)も計画されているようです。

1987年春には以前から仕様が公表されていたZ800が、Z280という名称でZilog社より製品発表されました。Z280はZ80のCPUを16ビット化したものであり、Z80とオブジェクトレベルのコンパチビリティがあります。キャッシュメモリを内蔵しており、現行市販されている16ビットよりもかなりの高性能を実現しています(図1・7)。

64180やZ280などの開発により、Z80系マイコンは8ビット標準マイコンの主流としての地位をここ当分保ち続けるものと思われます。

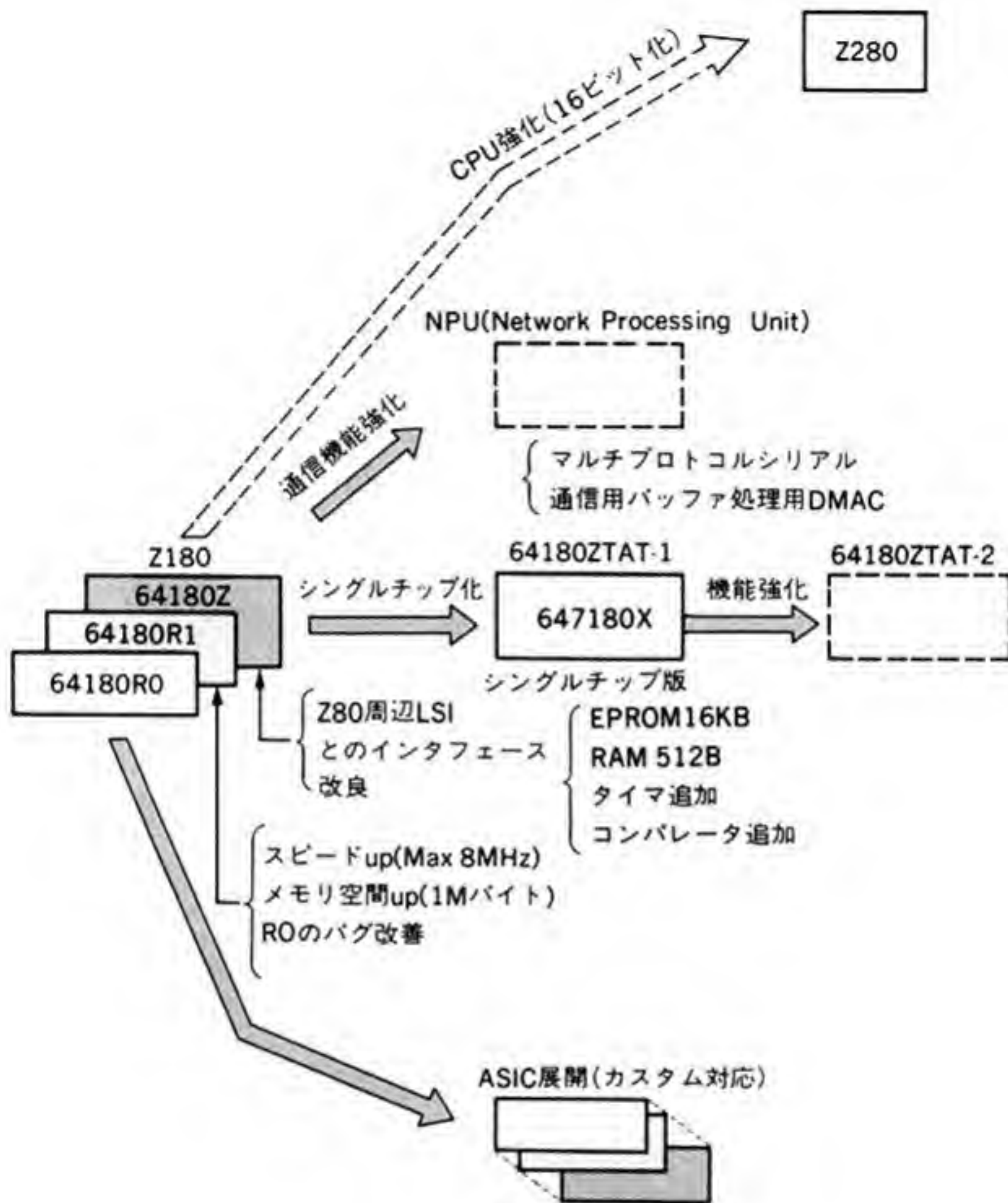


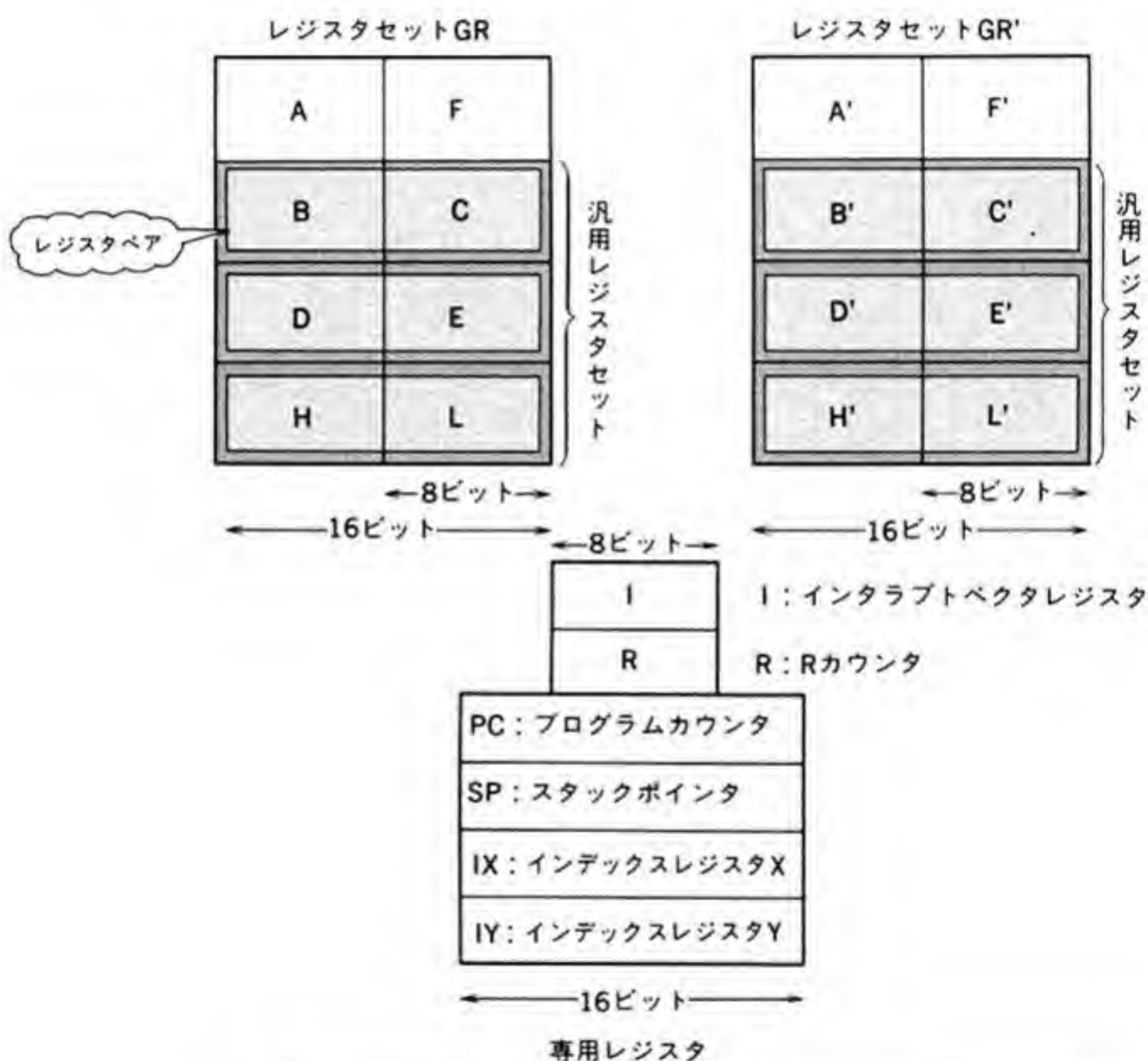
図 1・7 64180 ファミリーの展開

2. C P U アーキテクチャ

64180 は、Z80 に対し上位コンパチブルな CPU をもっています。全体的に、命令の実行ステート数は短くなっています。また、乗算命令やスリープ命令（低消費電力モードに設定）などが新規に追加されています。さらに、DRAM のリフレッシュはオペコードフェッチとは独立しています。このように、64180 の CPU は Z80 に対しパフォーマンスが高くなっています。

2・1 レジスタ構成

〔1〕 概 要 64180のCPUレジスタは、汎用レジスタセットと専用レジスタセットがあります。64180のプログラミングでは、これらのレジスタをどのように使いこなすかが重要になります（図2・1）。



通常、レジスタセットGRは2セットあり、一方の右肩にダッシュ「'」をつけて区別します。
16ビットのレジスタは「BC」「HL」「IX」「PC」のように2文字で表し、8ビットのレジスタは「A」「B」「C」「H」のように1文字で表します。

図 2・1 レジスタ構成

〔2〕汎用レジスタ 全く同一機能のレジスタセットが2組あります。汎用レジスタは8ビット長で、BとCレジスタ、DとEレジスタ、HとLレジスタにて、それぞれ16ビットのレジスタペアとしても使用できます。レジスタペアは、メモリやI/Oなどのアドレッシングレジスタとして使う場合が多いようです。また、アキュムレータAは算術論理演算などに使用する重要なレジスタです。フラグレジスタFの各ビットの値は、算術論理演算後のアキュムレータの内容に従って設定されます。汎用レジスタは、対応するレジスタ間でその内容を交換することができます。この操作は専用命令（EXX；EX AF, AF'）を使います。割込みやサブルーチンコール後のレジスタ退避に使用して効果があります。

〔3〕専用レジスタ 用途はあらかじめ決められています。PCは、次に実行される命令のアドレスを格納するプログラムカウンタです。SPは、割込みやサブルーチンコール後の復帰アドレスを格納する、スタック領域の先頭アドレスを示すスタックポインタです。さらに、IXとIYはインデックスアドレッシングや16ビット演算に使用するインデックスレジスタで、ともに同一機能です。Iレジスタは、割込みにおけるベクタ（16ビット）の上位8ビットを格納するインタラプトベクタレジスタです。Rカウンタはオペコードフェッチサイクルごとにインクリメントするカウンタです。RカウンタはZ80の場合とは異なり、リフレッシュアドレスとは無関係です。

PC、SP、IX、IYは16ビット長のレジスタで、IとRは8ビット長のレジスタです。

2・2 フラグレジスタ

〔1〕 概 要 フラグレジスタFは6つのフラグで構成されており、8ビットや16ビットの算術論理演算を実行した結果に従って値が設定されます。これらのフラグは、命令により「変化する」「変化しない」「不定」のどれかをとります。未定義の2ビットは常に不定です。

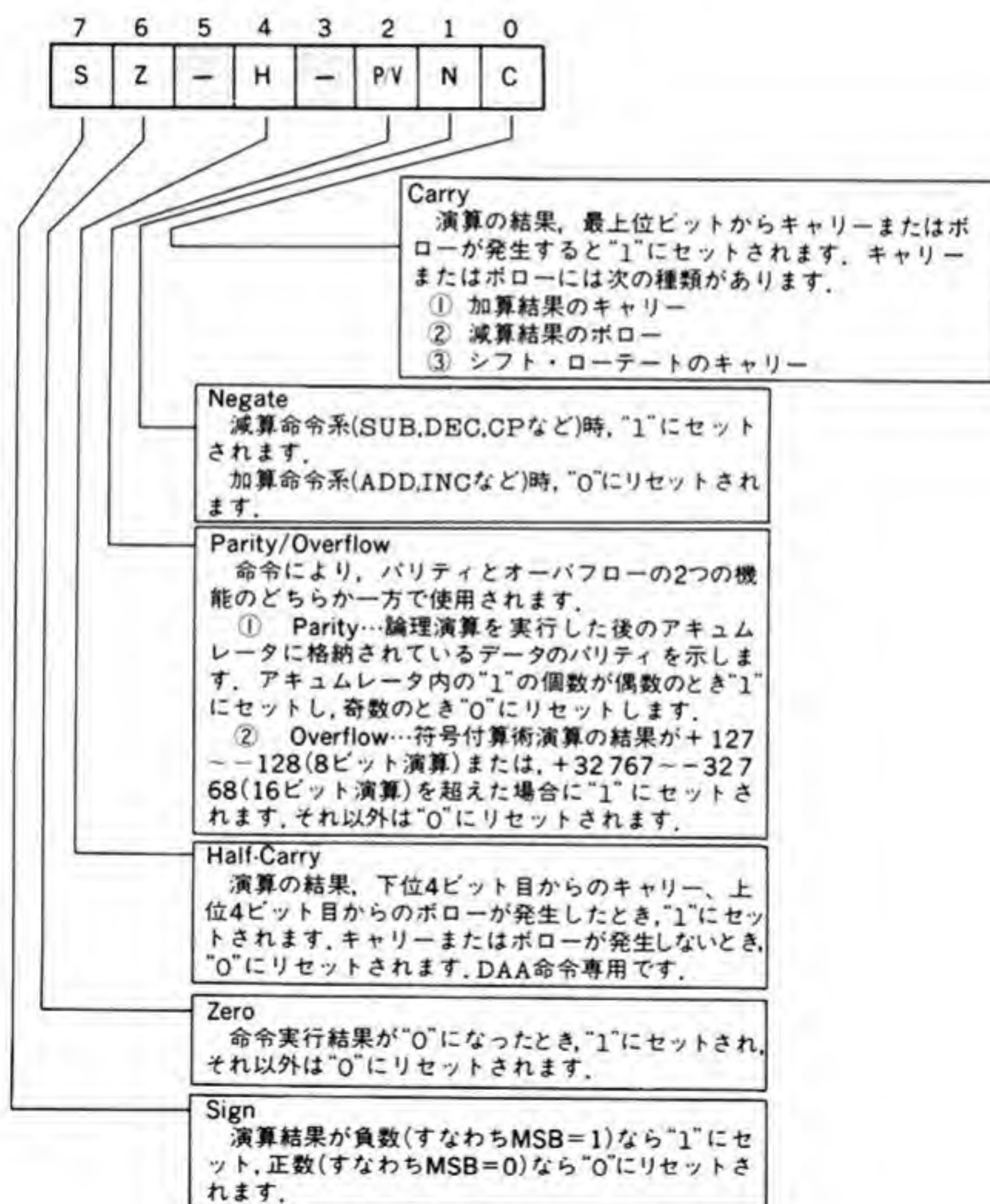


図 2・2 フラグレジスタ

〔2〕 **キャリー：C** 最上位ビットからのキャリーまたはボローにより“1”にセットされます。図2・2のようなケースでキャリーやボローが発生します。キャリーについては他のレジスタに影響を与えずにキャリーフラグを変化させる特殊な命令（CCF、SCF）があります。

〔3〕 **ネゲート：N** 減算命令系の実行時に“1”にセットされ、加算命令系の実行時に“0”にリセットされます。

〔4〕 **パリティ・オーバフロー：P/V** 命令ごとにパリティまたはオーバフローのどちらか一方の機能として働きます。パリティはアキュムレータ内の“1”の数が偶数のときに“1”にセット、奇数のときに“0”にリセットされます。オーバフローは、符号付算術演算の結果が定義されている範囲を超えたとき“1”にセットされます。

〔5〕 **ハーフキャリー：H** 下位4ビット目からのキャリーや、上位4ビット目からのボローが発生したとき、“1”にセットされます。10進演算の補正命令（DAA 命令）の実行時に使用します。

〔6〕 **ゼロ：Z** 命令の実行結果が0になったときに“1”にセットされます。また、ビットテスト命令でも影響を受けます。

〔7〕 **サイン：S** 演算結果が負の数ならば“1”にセットされます。

2.3 命令の構成

〔1〕 概 要 64180 の命令体系は Z80 上位コンパチブルであり、命令は1バイトから4バイトで構成されています。オペコードは、2バイトまたは3バイトに渡る命令をもつ点が特徴です。この場合、オペコードは EDH, CBH, DDH, FDH のいずれかが先頭にきます。

表 2.1 オペランドを伴わない命令の構成

バイト数	命令の構成	代 表 的 な 命 令		
1	オペコード	LD g, g'	01	g g'
2	第1 オペコード 第2 オペコード	SLP	EDH	76H

g, g' : 汎用レジスタ

表 2.2 オペランドを1バイト伴う命令の構成

バイト数	命令の構成	代 表 的 な 命 令		
2	オペコード オペランド	LD g, m	00	g 110 m
3	第1 オペコード 第2 オペコード オペランド	OUTO(m), g	EDH	00 g 001 m
		LD g, (IX+d)	DDH	01 g 110 d
4	第1 オペコード 第2 オペコード オペランド 第3 オペコード	RLC(IX+d)	DDH	CBH d 06H

g : 汎用レジスタ, m, n : イミディエイトデータ, d : ディスプレースメント

表 2・3 オペランドを2バイト伴う命令の構成

バイト数	命令の構成	代 表 的 な 命 令	
3	オペコード	LD(mn), A	32H
	第1 オペランド		n
	第2 オペランド		m
4	第1 オペコード	LD(IX+d), m	DDH
	第2 オペコード		36H
	第1 オペランド		d
	第2 オペランド		m
		LD(mn), IX	DDH
			22H
			n
			m

m, n: イミディエイトデータ, d: ディスプレースメント

〔2〕 **オペランドが伴わない命令の構成** 1バイトまたは2バイトの命令です。2バイト命令では、第1オペコードと第2オペコードがあります。たとえば表2・1のように、SLP命令ではEDHが第1オペコードで、76Hが第2オペコードです。また、LD g, g'命令のように、命令コードの中にレジスタを識別するための3ビットのフィールドをもっているものもあります。

〔3〕 **オペランドを伴う命令の構成** オペランドは1バイト（表2・2）または2バイト（表2・3）の長さであり、イミディエイトデータ m, n やディスプレースメント d となります。オペランドが2バイトの場合には2バイトのイミディエイトデータ mn または、ディスプレースメントとイミディエイトデータで構成されています。オペランドが2バイト伴う3バイト命令は、mnで構成され d はありません。

なお、オペランドが伴う命令でもLD g, mのように、オペコードの中にレジスタを識別するフィールドをもつ命令があります。

〔4〕 **インデックスレジスタを使った命令** 図2・3に示すように、HL または(HL)をオペランドとしてもつ命令のみ、HL→IXまたはIY, (HL)→(IX+

2. CPUアーキテクチャ

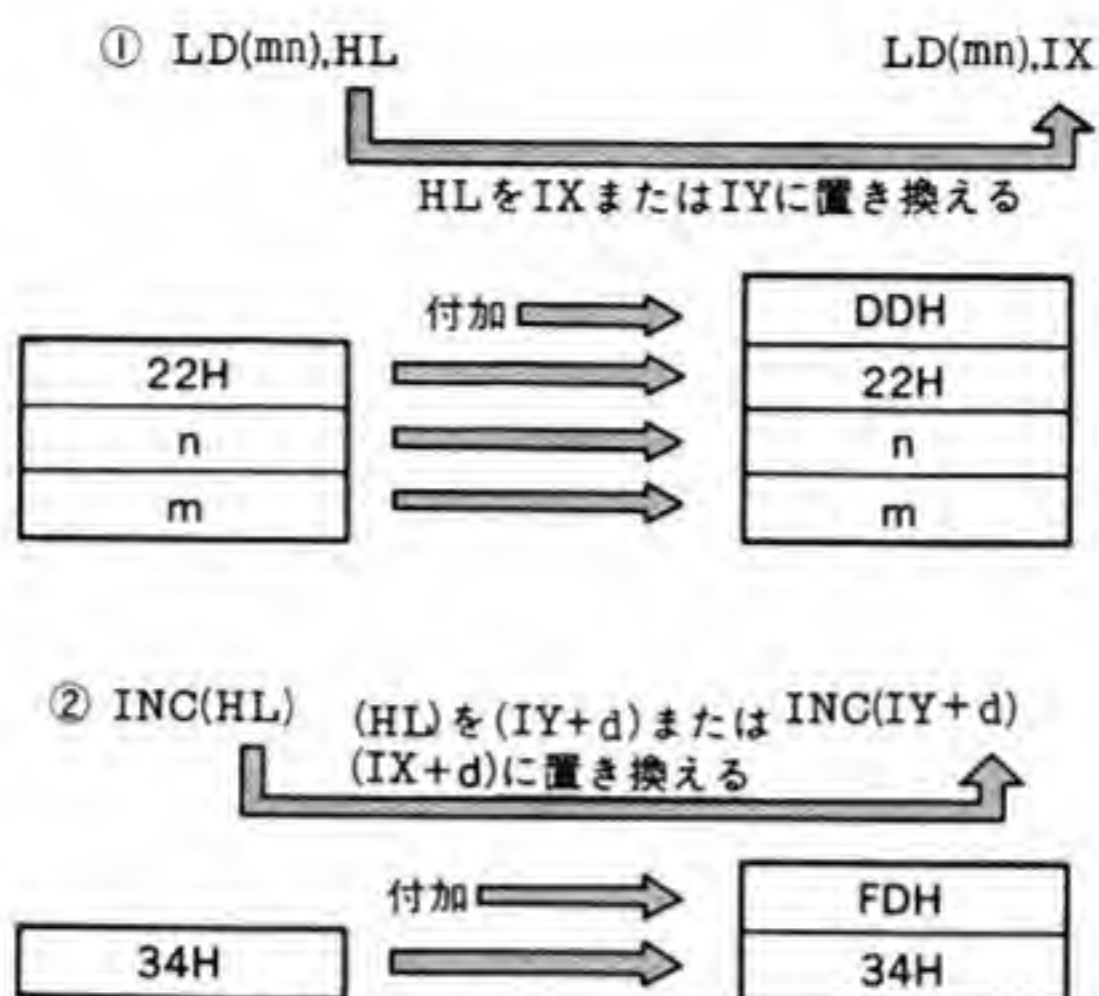


図 2・3 インデックスを使った命令の構成

d) または $(IY+d)$ と置き換えると、同じオペレーションでインデックスレジスタを使った命令となります。オペコードは IX の場合先頭に DDH を、IY の場合は FDH を付加します。ただし例外もあり、JP(HL) の場合は HL→IX または IY に置き換えます。また、「EX DE, HL」の場合は置き換えると未定義コードとなります。

〔5〕 **未定義コード** 第1 オペコードには未定義コードがありませんが、第2 オペコードまたは第3 オペコードには未定義コードがあります。64180 では、未定義コードを CPU がフェッチすると、TRAP 割込みが発生します。詳細は、「9・7 ノンマスカブルな割込み」を参照してください。

2・4 データの配置

64180 の命令では、転送命令やプログラム制御命令などで指定されるメモリやレジスタへの 8 ビットデータの配置には、次のような場合があります。

〔1〕 レジスタとイミディエイトデータ 図 2・4 のように、オペコードの次に配置されたイミディエイトデータ n が、レジスタペアや 16 ビットレジスタの下位 8 ビットに対応します。上位 8 ビットは m が対応します。

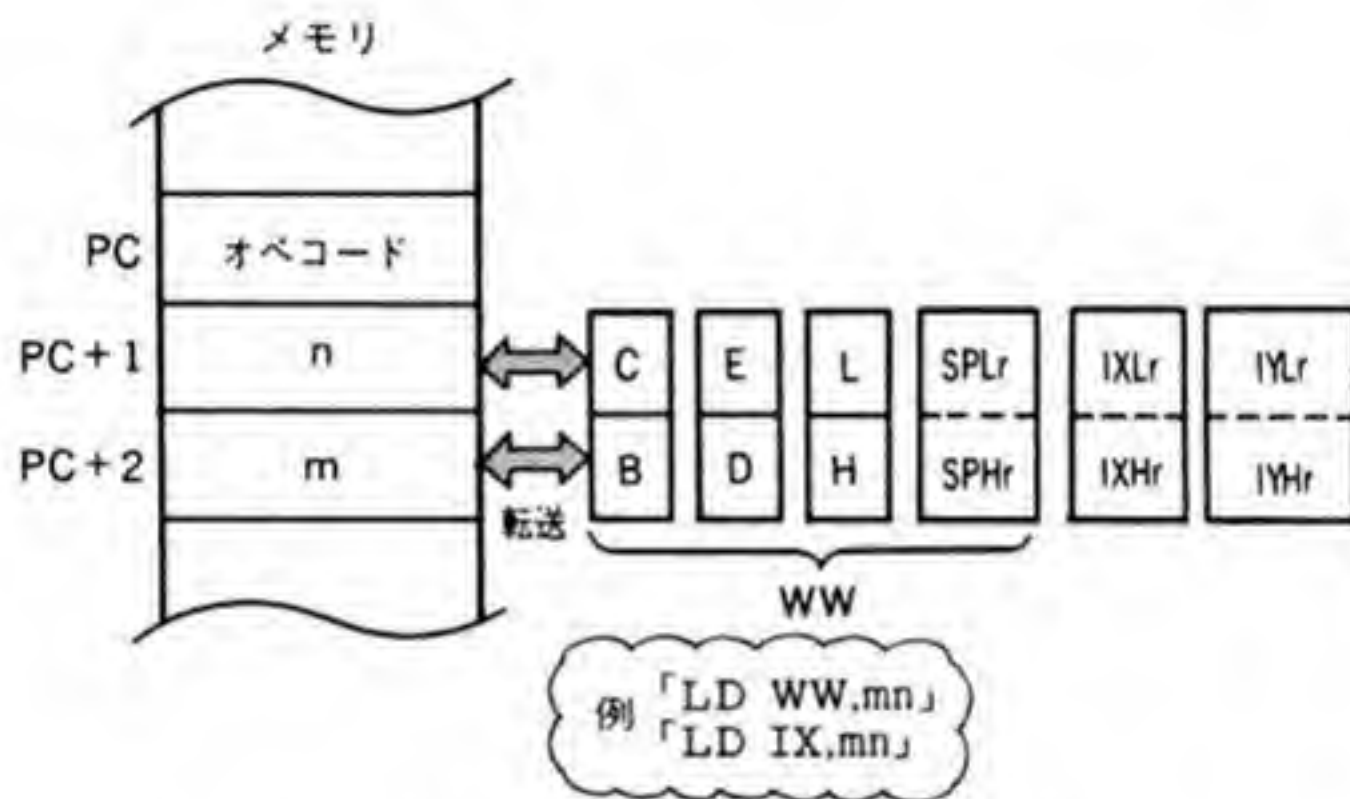


図 2・4 レジスタとイミディエイトデータ mn との対応関係

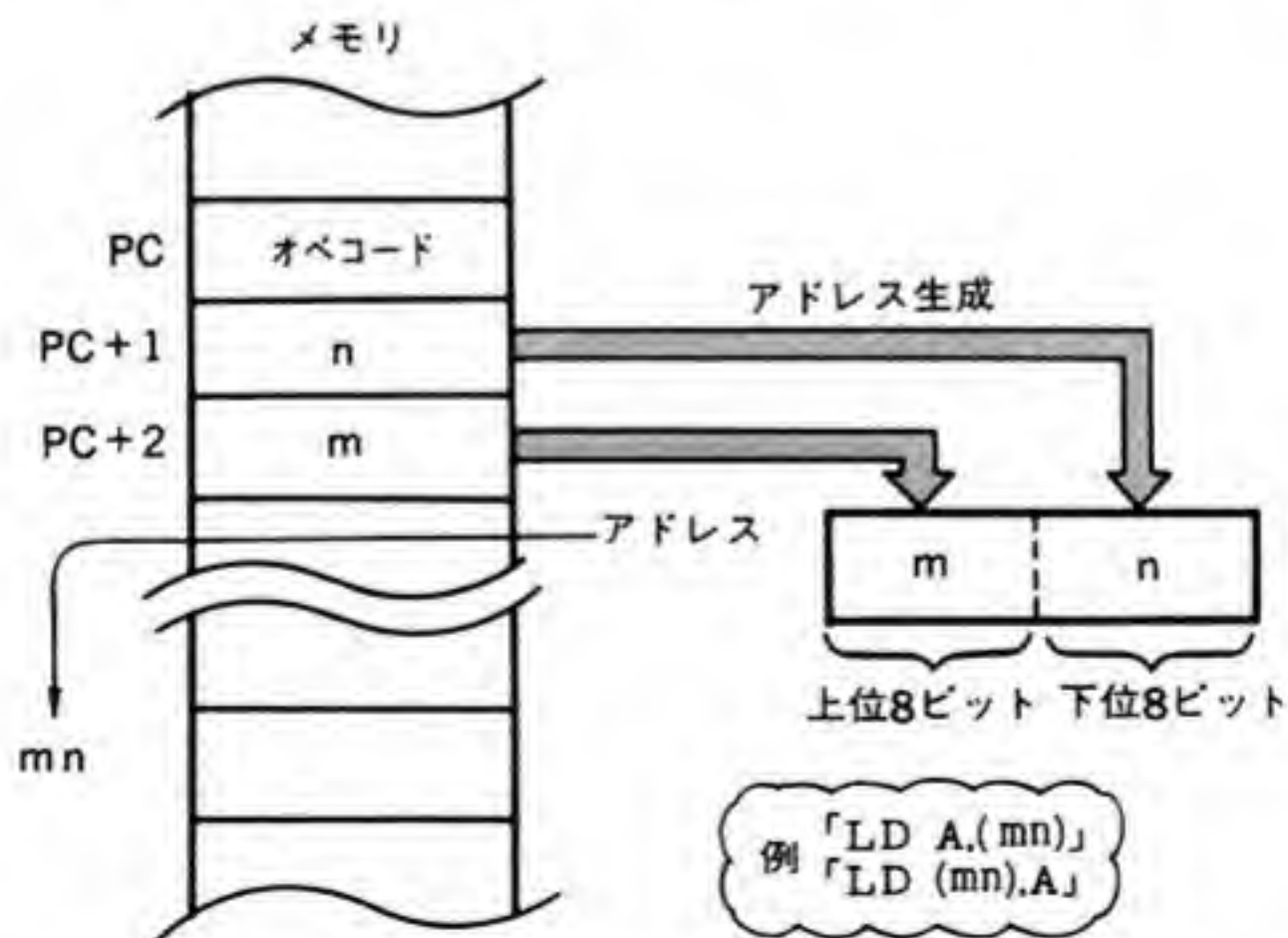


図 2・5 イミディエイトデータによるアドレスの生成

2. CPU アーキテクチャ

〔2〕 **アドレス生成** イミディエイトデータ **mn** によるアドレス生成です。図 2・5 のように、アドレスの上位 8 ビットは **m** によって、下位 8 ビットは **n** によって生成されます。

〔3〕 **レジスタと間接アドレッシングによるメモリデータとの対応** LD IX, (mn) 命令時です。アドレス生成は図 2・5 と同じであり、そのアドレスで示されるメモリとレジスタとの対応は図 2・4 と同等です。ただし、この時 PC は **mn** に置き換えます。

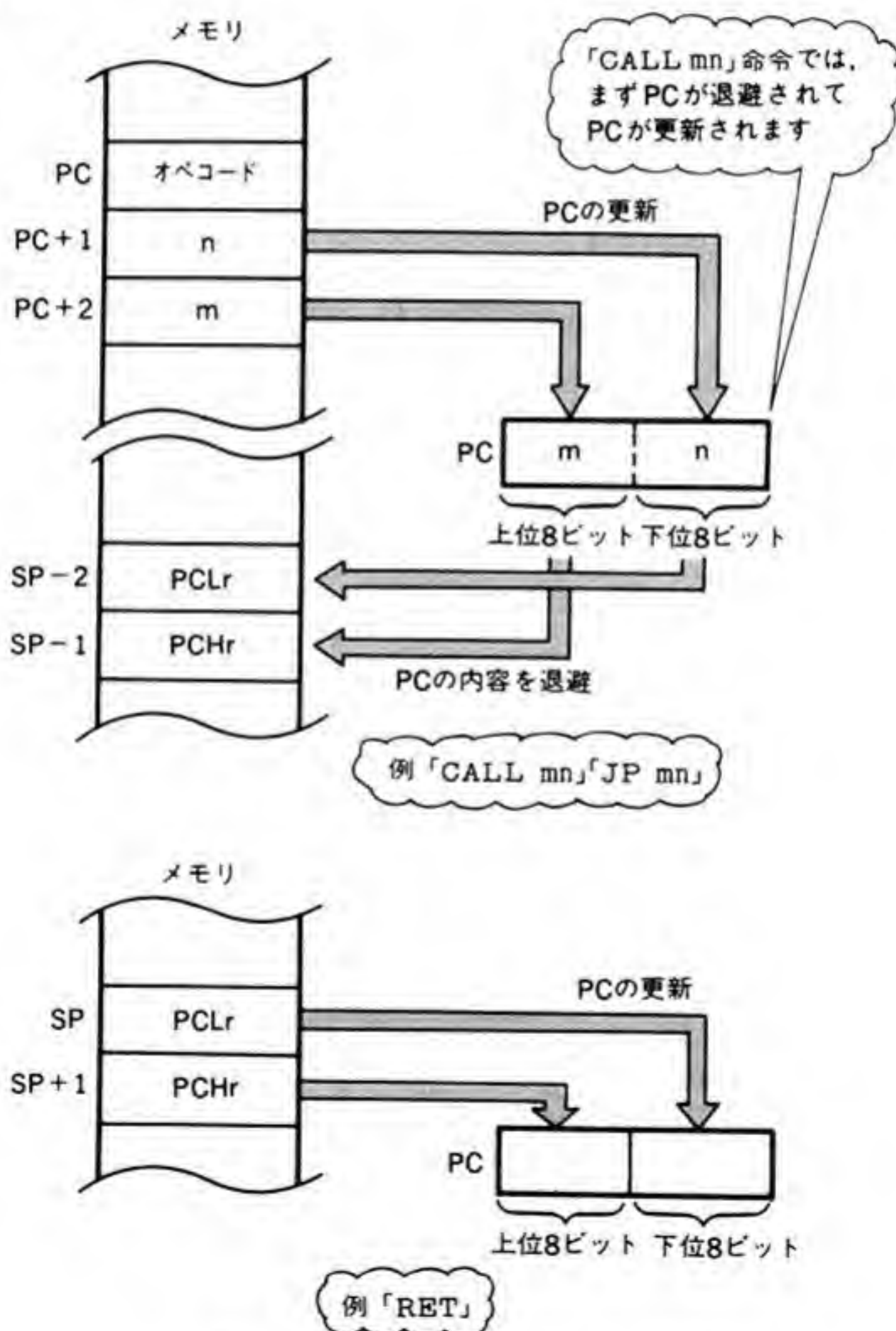


図 2・6 プログラムカウンタ PC の更新

〔4〕 プログラムカウンタ PC の更新 図2・6に示します。PCの下位8ビットには、「CALL mn」「JP mn」命令の場合はイミディエイトデータ n が、「RET」命令の場合はスタックポインタ SP で示されるスタックの内容が転送されます。上位8ビットにはそれぞれの命令で、イミディエイトデータ m または $SP+1$ で示されるスタックの内容が転送されます。「CALL mn」命令では、その前に PC の内容がスタックへ退避されます。

〔5〕 レジスタとスタックとの対応 図2・7に示します。レジスタペアや16ビットレジスタの上位8ビットは、PUSH 命令では $SP-1$ で示されるスタックの内容が、POP 命令や交換命令では $SP+1$ で示されてスタックの内容が対応します。下位8ビットは、それぞれ $SP-2$ と SP で示されるスタックの内容が対応します。

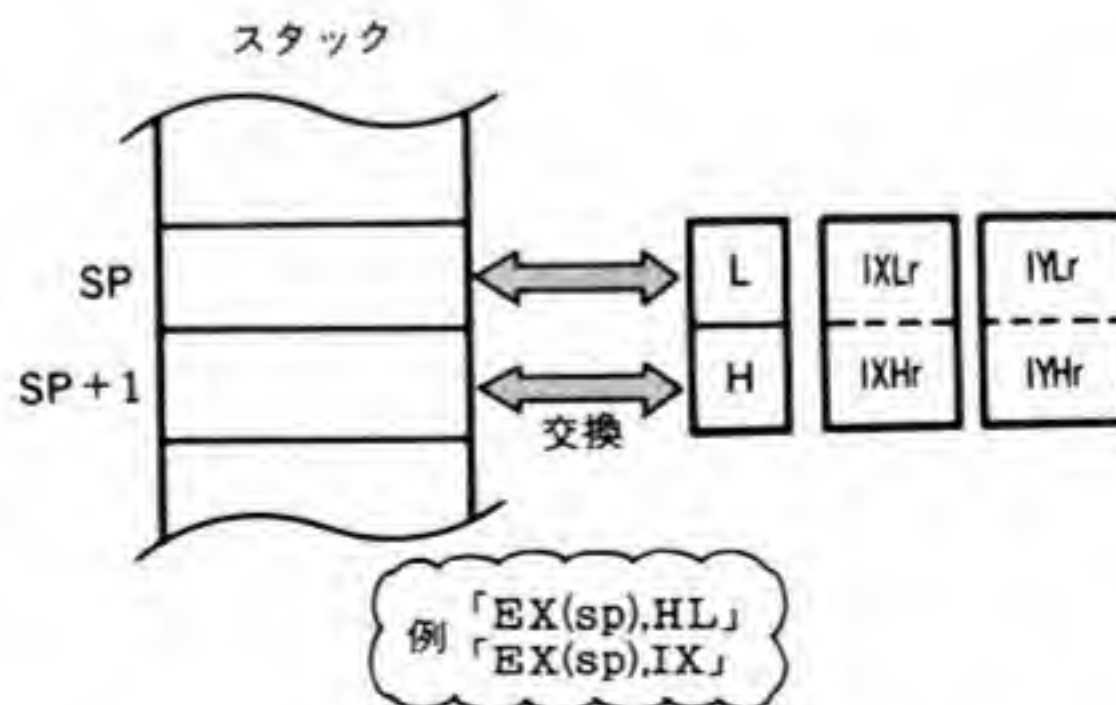
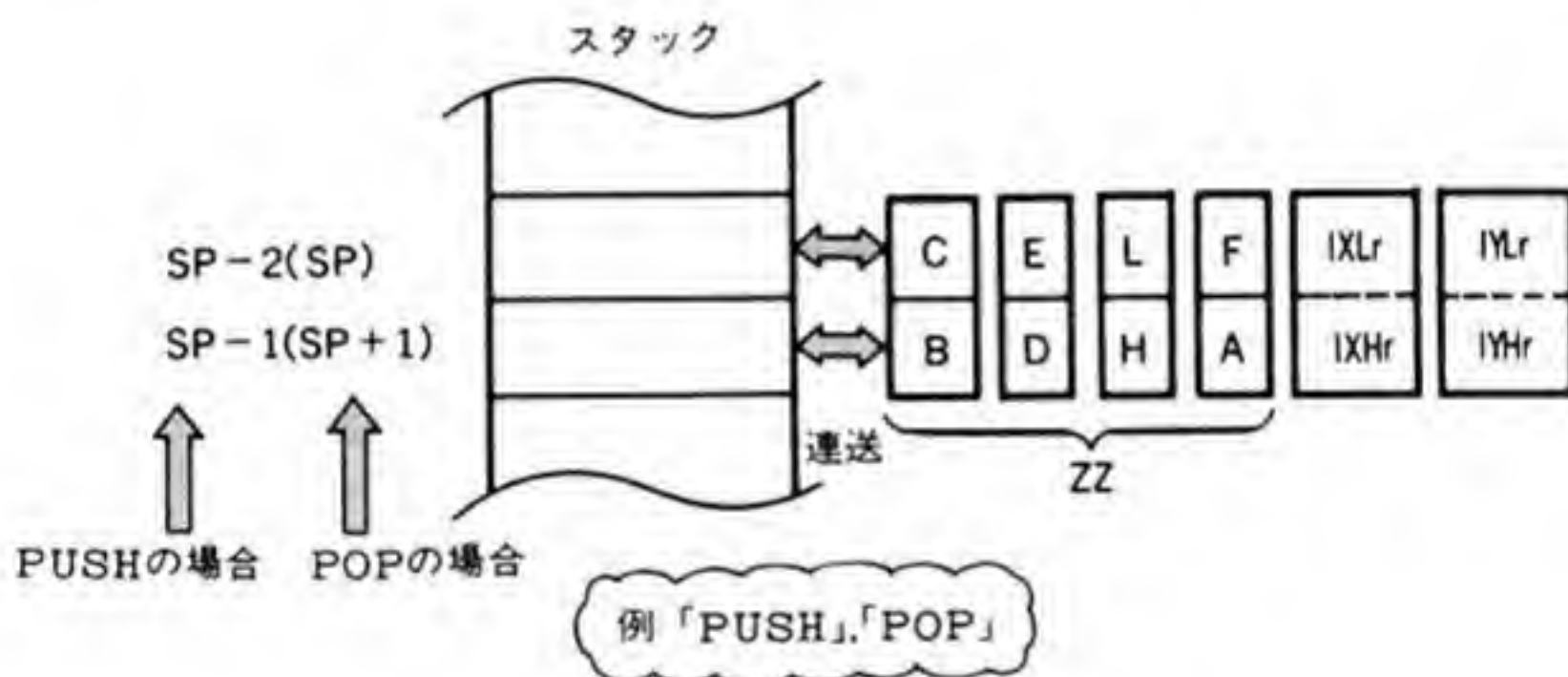


図 2・7 スタックの内容とレジスタとの転送または交換

2.5 アドレッシングモード

〔1〕 **インプライド (IMP)** オペコード中に含まれる情報に基づくレジスタアドレッシングモードです。たとえば、DAA, CPL, RLA 命令などが相当します。

〔2〕 **レジスタダイレクト (REG)** オペコード中の 2 ビットまたは 3 ビットで表されるフィールドによって、8 ビット、16 ビットのレジスタが指定されるアドレッシングモードです。「LD B, C」, 「INC B」, 「MLT BC」命令などが相当します。

〔3〕 **レジスタインダイレクト (REGI)** 図 2・8 にこのアドレッシングモードを示します。汎用レジスタを 16 ビットのアドレスレジスタとして使用し、これによってメモリ上のオペランドを指定するアドレッシングモードです。「DEC(HL)」, 「RL(HL)」, 「LDD」命令などが相当します。

〔4〕 **インデックス (INDX)** 図 2・9 にこのアドレッシングモードを示します。インデックスレジスタ (IX, IY) とディスプレースメント (d) との加算によって実効アドレスを生成するアドレッシングモードです。「RL(IX+d)」, 「DEC(IX+d)」, 「SLA(IY+d)」命令などが相当します。

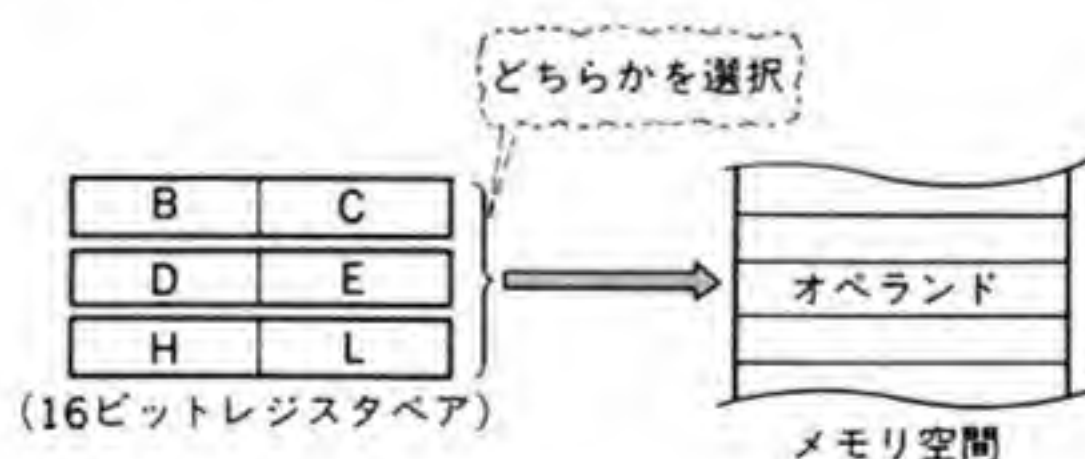


図 2・8 REGI アドレッシング

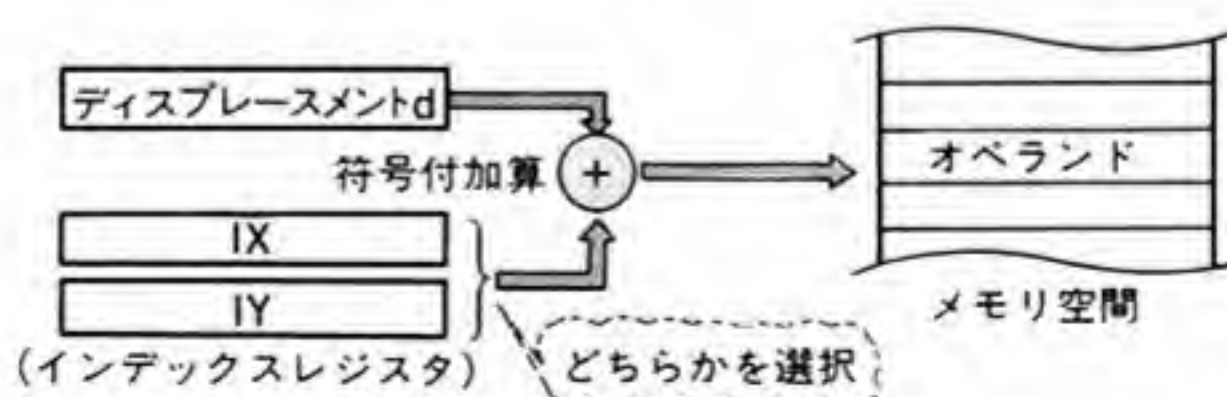


図 2・9 INDX アドレッシング

2・5 アドレッシングモード

〔5〕 **エクテンド (EXT)** 図2・10にこのアドレッシングモードを示します。命令の2バイトを16ビットのアドレスとして、直接メモリ上のオペランドを指定するアドレッシングモードです。「CALL」、「JP」命令などが相当します。

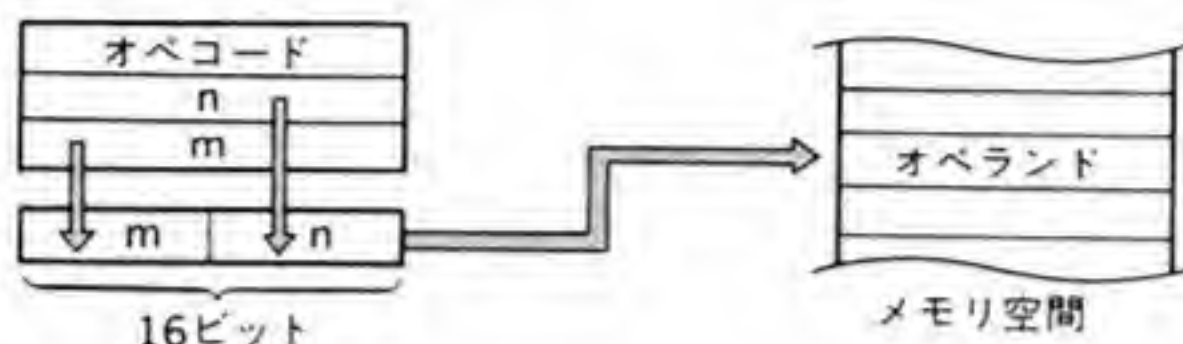


図 2・10 EXT アドレッシング

〔6〕 **イミディエイト (IMMED)** 図2・11にこのアドレッシングモードを示します。命令中の1バイトまたは2バイトを直接オペランドとして使用するアドレッシングモードです。「LD A, 00H」、「LD HL, 55AAH」などが相当します。

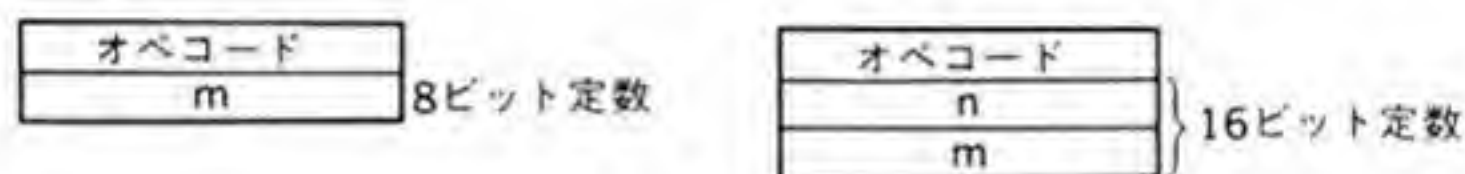


図 2・11 IMMEDIATE アドレッシング

〔7〕 **リラティブ (REL)** 図2・12にこのアドレッシングモードを示します。プログラムカウンタにディスプレースメントを加算して、ブランチアドレスを生成するアドレッシングモードです。「JR」、「DJNZ」命令などが相当します。



図 2・12 REL アドレッシング

〔8〕 **IO** I/O 命令においてのみ使用されるアドレッシングモードです。

2・6 メモリ空間とI/O空間

〔1〕 概 要 64180 では図 2・13 のように、データをアクセスする空間はメモリ空間と I/O 空間の 2 つがあります。CPU が直接指定できるメモリ空間は $2^{16} = 64 \text{ K}$ バイトであり、I/O 空間も 64K バイトです。また、64180 は MMU (Memory Management Unit) を内蔵しているため、MMU を通じてメモリ空間は最大 1 M バイトに拡張できます。

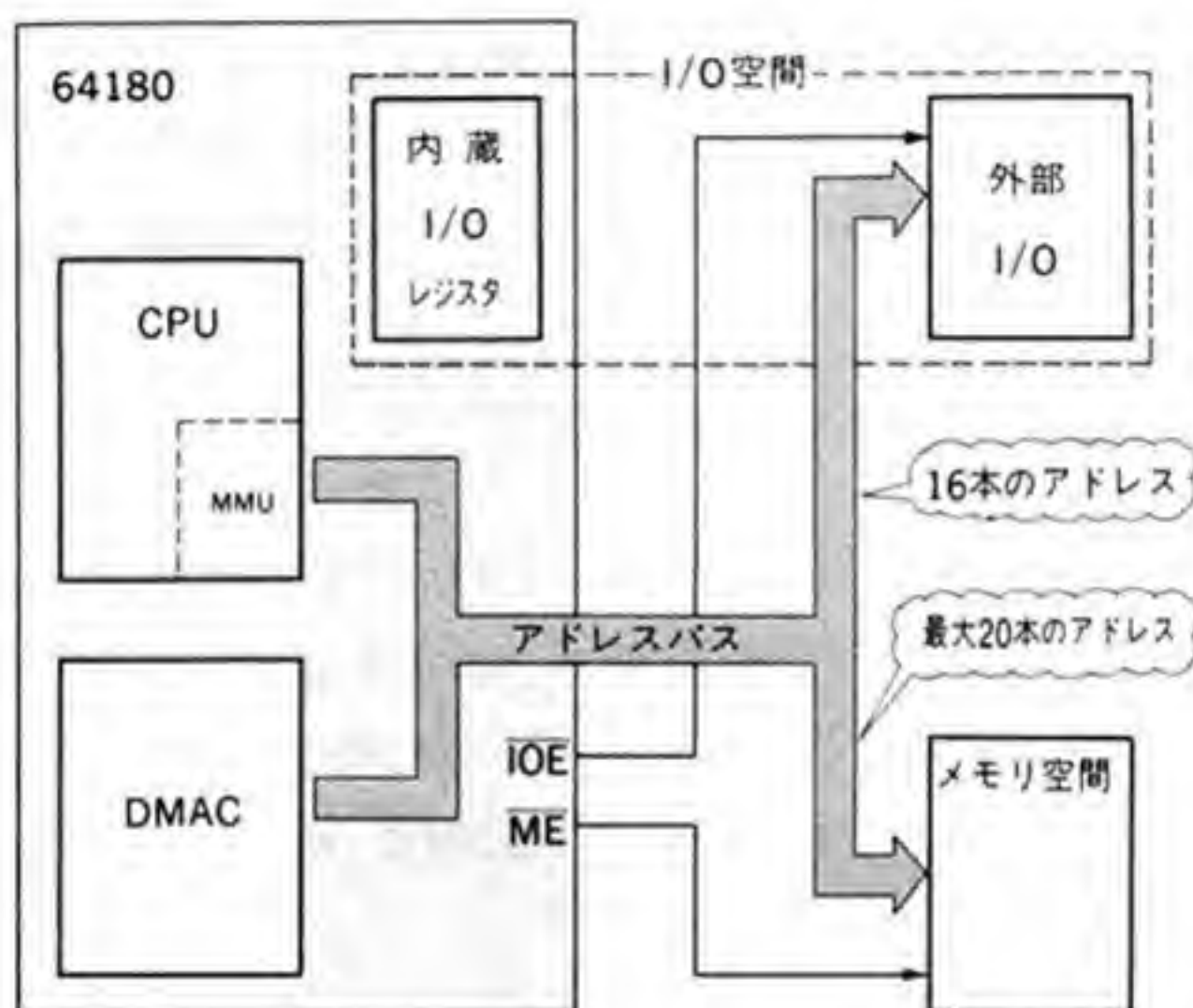


図 2・13 メモリ空間・I/O空間

〔2〕 メモリ空間 メモリ空間をアクセスする場合、制御信号の $\overline{\text{ME}}$ が “L” になります。メモリ空間へのアクセスは、CPU または DMAC (Direct Memory Access Controller) が行います。CPU では、「LD (HL), A」や「PUSH BC」「POP BC」命令などの通常の命令にて行われます。DMAC では、メモリ空間をアクセスするか I/O 空間をアクセスするかを内蔵レジスタで設定し、メモリ空間が選択されると、CPU を介せずに直接アクセスされます。DMAC がメモリ空間をアクセスする場合は、CPU と同じように $\overline{\text{ME}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ などの制御信号を出力します。

〔3〕 I/O 空間 I/O 空間をアクセスする場合、制御信号の $\overline{\text{IOE}}$ が

“L” になります。I/O 空間へのアクセスも、CPU または DMAC が行います。CPU では、I/O 空間をアクセスするための特別な命令（IN/OUT 命令）にて行います。DMAC では、内蔵 I/O レジスタで I/O 空間のアクセスを設定し、I/O 空間が選択されると CPU を介せずに直接アクセスされます。

I/O 空間の一部は図 2・13 の点線で示すように、内蔵の I/O レジスタがマッピングされています。そのアドレスについては付録を参照してください。なお、この内蔵 I/O レジスタのアドレスは上位 8 ビットは 00H に固定されており、下位 8 ビットは次のようになっています。ビット 0 からビット 5 は各レジスタに固有の値であり、ビット 6 とビット 7 は、I/O コントロールレジスタの IOA6 と IOA7 で設定します。したがって、図 2・14 に示すように 64 バイト単位で 4 か所に移動できます。

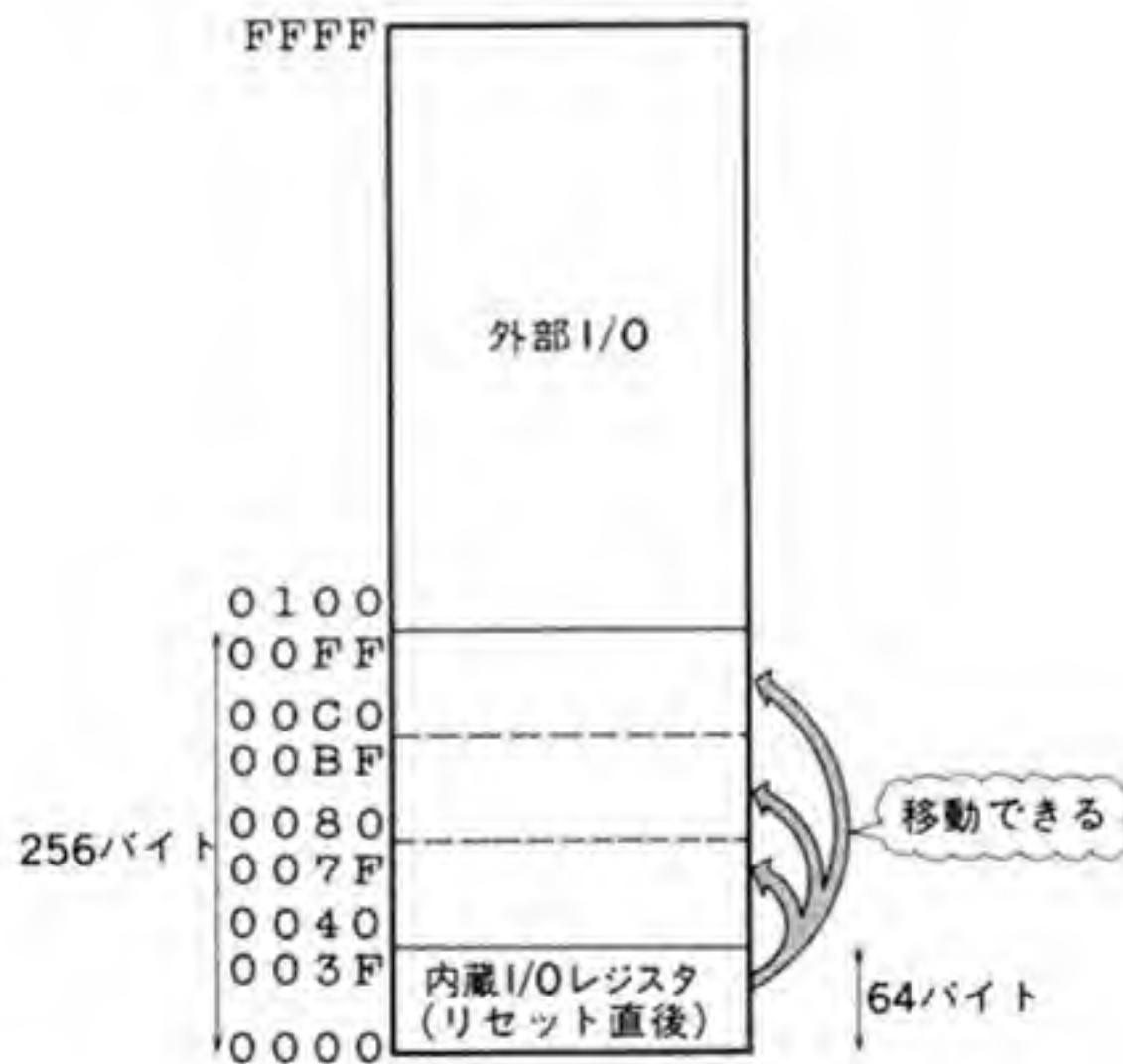


図 2・14 I/O アドレスマップ

2・7 命令の分類

64180の命令は、表2・4に示す5種類に大別できます。

〔1〕演算命令 メモリや8ビットまたは16ビットレジスタの数学的な論理演算を行う命令です。加算、減算、8ビット乗算、論理積、論理和、排他

表2・4 命令の分類

No.	大分類	中分類	小分類
1	演算命令	算術論理演算 (8ビット)	ADD, ADC, SUB, SUBC, AND, OR, XOR, CP, INC, DEC, TST, CPL, MLT, NEG
		ローテート およびシフト	RLA, RL, RLCA, RLC, RRA, RR, RRCA, RRC, SLA, SRA, SRL, RLD, RRD
		ビット操作	SET, RES, BIT
		算術演算 (16ビット)	ADD, ADC, INC, DEC, SBC
2	転送命令	8ビット転送	レジスタ → レジスタ レジスタ → メモリ
		16ビット転送	レジスタ → レジスタ レジスタ → メモリ
		ブロック転送	LDD, LDI, LDDR, LDIR
		ブロックサーチ	CPD, CPI, CPDR, CPIR
		スタック処理	PUSH, POP
		交換	EX, EXX
3	プログラム 制御命令	—	CALL, JP, JR, DJNZ, RET, RETI, RETN, RST
4	I/O命令	入力	IN, INO, TSTIO
		出力	OUT, OUTO
		ブロック入力	IND, INI, INDR, INIR
		ブロック出力	OTDM, OUTI, OTIM, OUTD, OTDMR, OTDR, OTIR, OTIMR
5	特殊制御	演算	DAA, SCF, CCF
		割り込み	EI, DI, IM
		動作モード	HALT, SLP
		その他	NOP

的論理和、インクリメント、デクリメント、コンペア、ローテートやシフト、ビットセットやリセット、ビットテストなどがあります。加算と減算、ローテートとシフトにはキャリーとともに実行する命令があります。

〔2〕 **転送命令** 8ビットまたは16ビットのレジスタやメモリの内容を転送する最も基本的な命令です。単なるデータ転送のほかに、汎用レジスタをいくつか使って複数バイト転送するブロック転送命令やブロック比較命令、レジスタをスタックへ退避したり復帰したりする命令、レジスタとレジスタまたはスタックとの交換命令があります。

〔3〕 **プログラム制御命令** プログラムの流れを制御する命令です。プログラムを分岐するジャンプ命令とサブルーチンコール命令、サブルーチンや割込み処理ルーチンからの復帰命令があります。また、特定のアドレスへ強制的に分岐するリスタート命令もあります。

〔4〕 **I/O 命令** I/O とのデータの読出し・書込みを行う命令です。I/O 空間の上位8ビットが常に OOH とした命令もあり、64180 の内蔵 I/O レジスタのアクセスに使います。

〔5〕 **特殊制御命令** 1～4 以外の命令で、10 進補正、キャリー制御、割込み制御、およびホルトやスリープモード選択命令などがあります。

3. C P U タイミング

64180は「システム集積型」マイコンですが、ROMやRAMなどのメモリを内蔵していません。また、内蔵I/O機能だけでは不足する場合があります。そのため、外部にメモリやI/O機器を接続します。そこで、64180はこれらを制御するための信号をもっており、そのタイミングは各種メモリや周辺LSIとのインタフェースが容易となるようにZ80から変更されています。Z80周辺LSIについては、64180Z版にてインタフェースが容易となるように工夫されています。

3・1 メモリリードサイクル・ライトサイクル

〔1〕 概 要 メモリ空間へアクセスするときのサイクルで、1サイクルは T_1 、 T_2 、 T_3 ステートで構成されています。低速メモリのアクセスの場合は、ウェイトステートとして T_w が T_2 と T_3 の間に挿入されることがあります。

〔2〕 メモリリードサイクル オペコードフェッチ以外のメモリリードサイクル（図3・1）です。そのシーケンスは次のとおりです。① T_1 ステートの前半でプログラムカウンタ PC の内容がアドレスバスに出力されます。② T_1 ステートの後半で、 \overline{ME} と \overline{RD} がアクティブになりメモリへの起動がかかります。③ データバス上にのったメモリリードデータを、 T_3 ステートの ϕ クロックの立下りでCPUは取り込みます。そのとき、 \overline{ME} と \overline{RD} は T_3 ステートの後半でインアクティブとなります。④ アドレスバスは、次のサイクルの先頭までそのアドレスを保持しています。

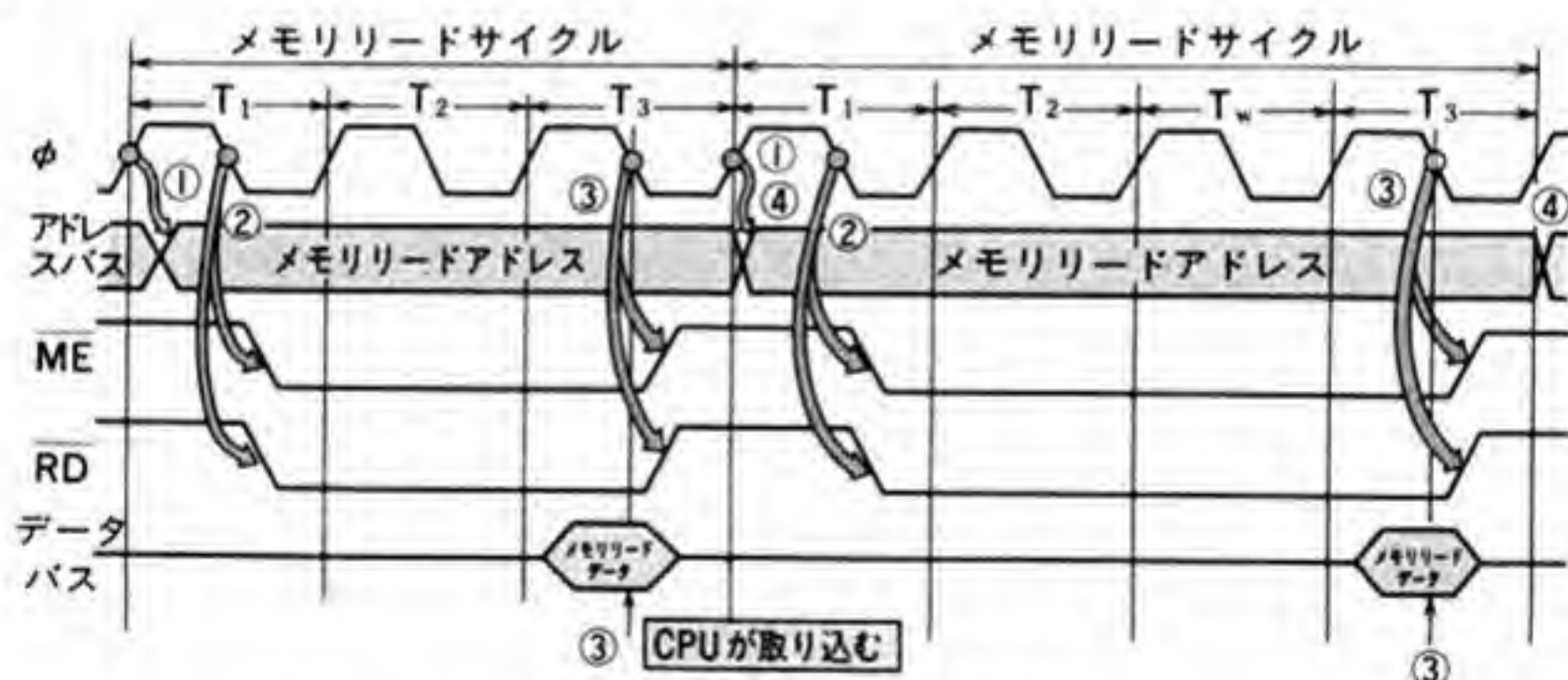


図 3・1 メモリリードサイクル

〔3〕 メモリライトサイクル このサイクル（図3・2）のシーケンスは次のとおりです。① アドレスは T_1 ステートの前半で出力されます。② T_1 ステートの後半で、 \overline{ME} をアクティブにすると同時にメモリライトデータが出力されます。③ T_2 ステートの前半で \overline{WR} がアクティブとなって、メモリに起動がかかります。④ \overline{ME} と \overline{WR} は T_3 ステートの後半でインアクティブとなりますが、メモリライトデータは次のバスサイクルの先頭まで保持されています。

Z80 では、 \overline{WR} がアクティブになるタイミングは T_2 ステートの後半です。64180

3・1 メモリリードサイクル・ライトサイクル

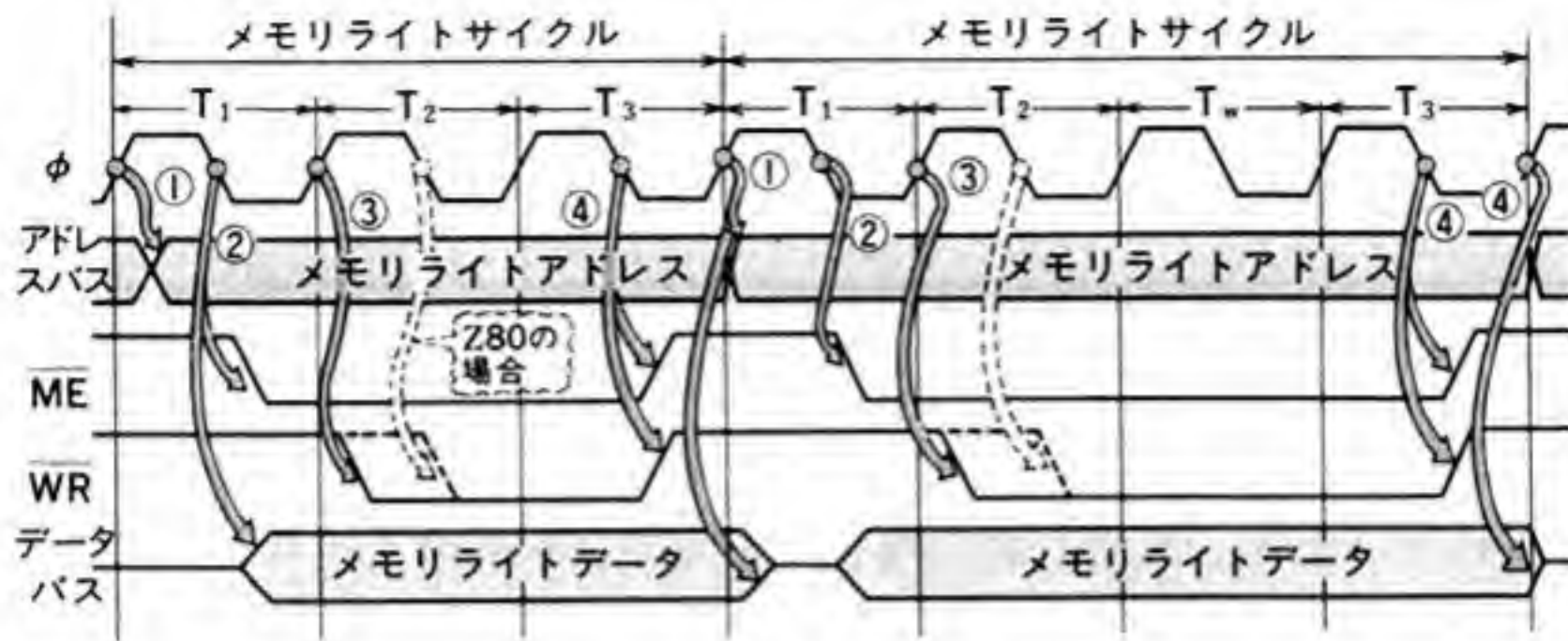


図 3・2 メモリライトサイクル

は、半クロック速くアクティブとなります。

〔4〕 **ウェイトステートが挿入されている場合** ウェイトステート T_w は、 T_2 ステートと全く同一動作をします。したがって、アドレス、 \overline{ME} 、 \overline{RD} 、 \overline{WR} などは、 T_2 ステートから引き延ばされています。 T_w ステートの挿入法は、端子要求によるものとソフトウェアによるものとがあります。

3・2 I/Oリードサイクル・ライトサイクル

〔1〕 概 要 内蔵 I/O レジスタや外部 I/O 空間へのアクセス時のサイクルです。外部空間へのアクセス時では、1 サイクルが T_1 、 T_2 、 T_w 、 T_3 ステートで構成されています。低速 I/O の場合、さらに T_w が挿入されることがあります。

〔2〕 I/O リードサイクル このサイクル（図 3・3）のシーケンスは次のとおりです。① T_1 ステートの前半で、アドレスがアドレスバスに出力されます、② T_1 ステートの後半で、 \overline{IOE} と \overline{RD} がアクティブとなって I/O に起動をかけます、③ T_3 ステートの ϕ クロックの立下りで、CPU はリードデータを取り込みます、それと同時に、 \overline{IOE} と \overline{RD} は T_3 ステートの後半でインアクティブとなり

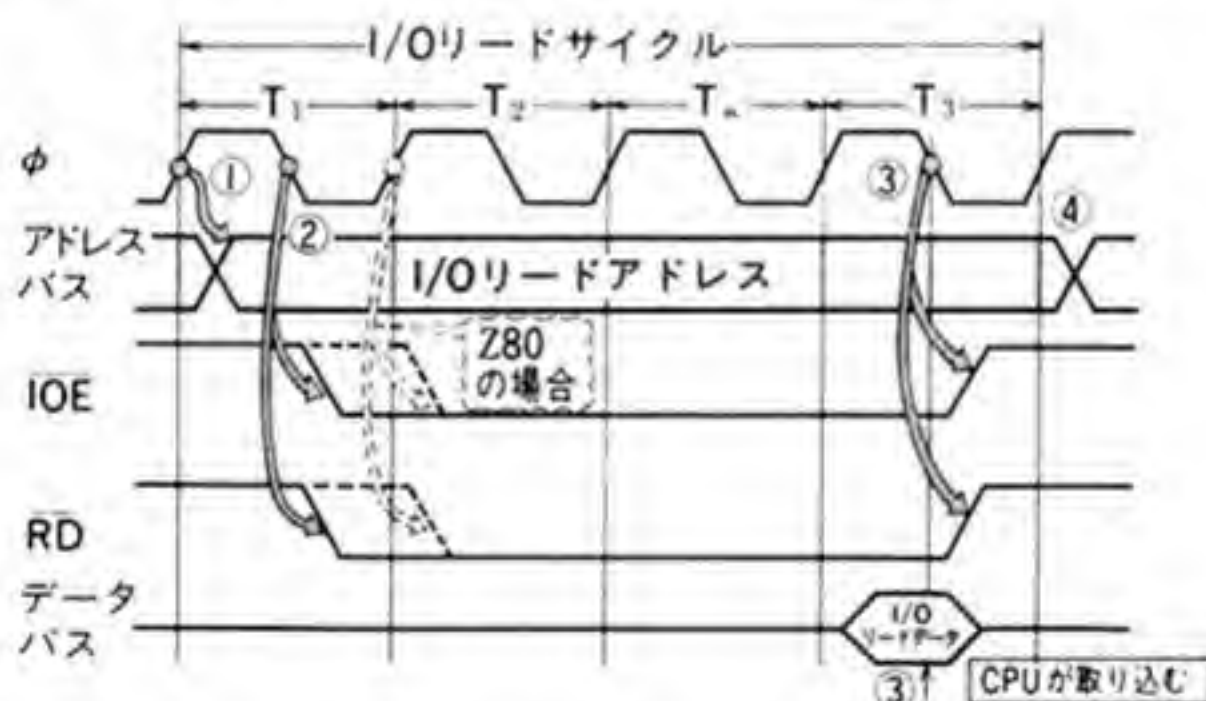


図 3・3 I/O リードサイクル

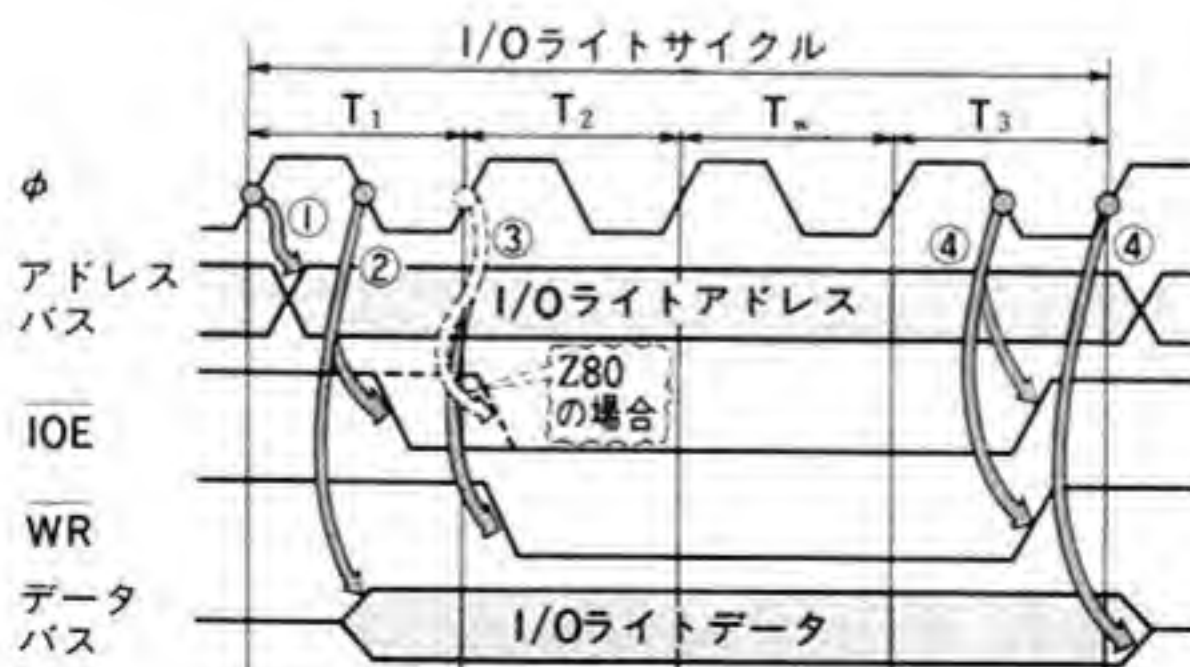


図 3・4 I/O ライトサイクル

3・2 I/O リードサイクル・ライトサイクル

ます、④アドレスは次のバスサイクルの先頭まで保持されています。

Z80 では、 $\overline{\text{IORQ}}$ と $\overline{\text{RD}}$ のアクティブタイミングは T_2 ステートの前半ですので、64180 は半クロック速くアクティブとなります。

〔3〕 **I/O ライトサイクル** このサイクル (図 3・4) のシーケンスは次のとおりです。① T_1 ステートの前半で、アドレスがアドレスバスに出力されます、② T_1 ステートの後半で $\overline{\text{IOE}}$ はアクティブとなり、同時に I/O ライトデータが出力されます、③ T_2 ステートの前半で $\overline{\text{WR}}$ はアクティブとなり、I/O に起動がかかります、④ ライトデータは次のバスサイクルまで保持されていますが、 $\overline{\text{IOE}}$ と $\overline{\text{WR}}$ は T_3 ステートの後半でインアクティブとなります。

Z80 では、 $\overline{\text{IORQ}}$ のアクティブタイミングは T_2 ステートの前半ですので、64180 は半クロック速くアクティブとなります。

〔4〕 **ウェイトステート挿入の場合** これらのサイクルでも、メモリリード・ライトサイクル同様に、 T_w をさらに挿入することができます。

〔5〕 **64180Z 版の場合** $\overline{\text{IOE}}$ と $\overline{\text{RD}}$ のアクティブタイミングを、ソフトウェアで T_2 の前半に変更することができます。内蔵の動作モードコントロールレジスタの $\overline{\text{IOC}}$ ビットを“0”に設定します。Z80 周辺 LSI とのインタフェースに便利です (図 3・5、図 3・6)。

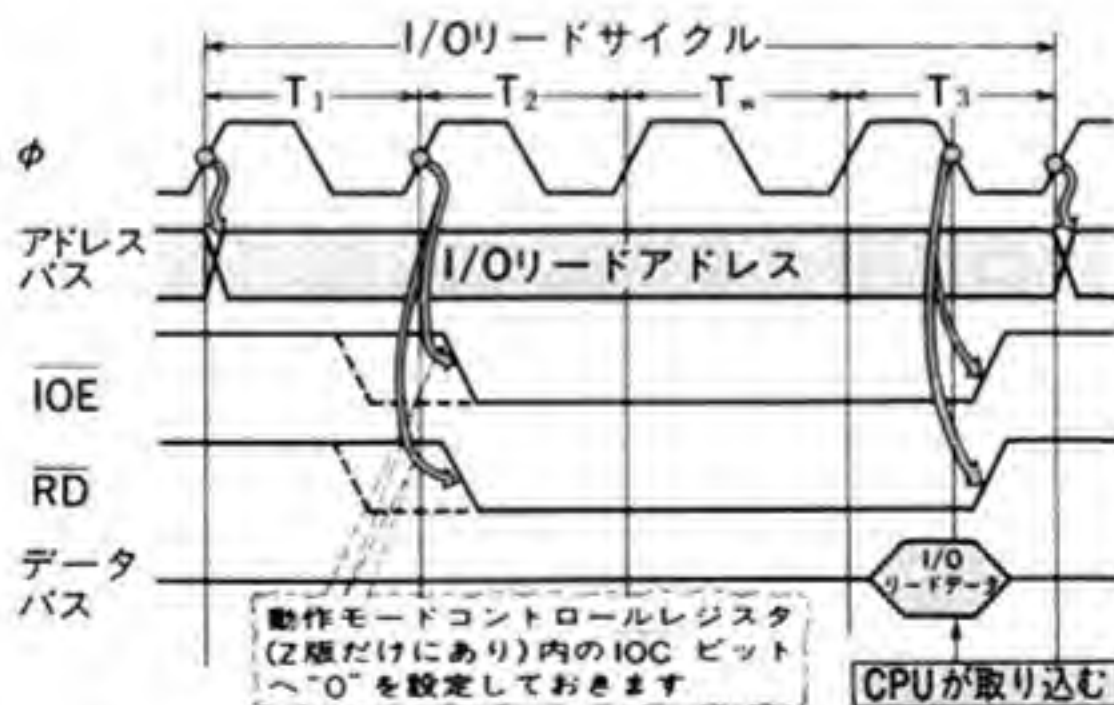


図 3・5 I/O リードサイクル ($\overline{\text{IOC}} = "0"$ の場合)

3. CPU タイミング

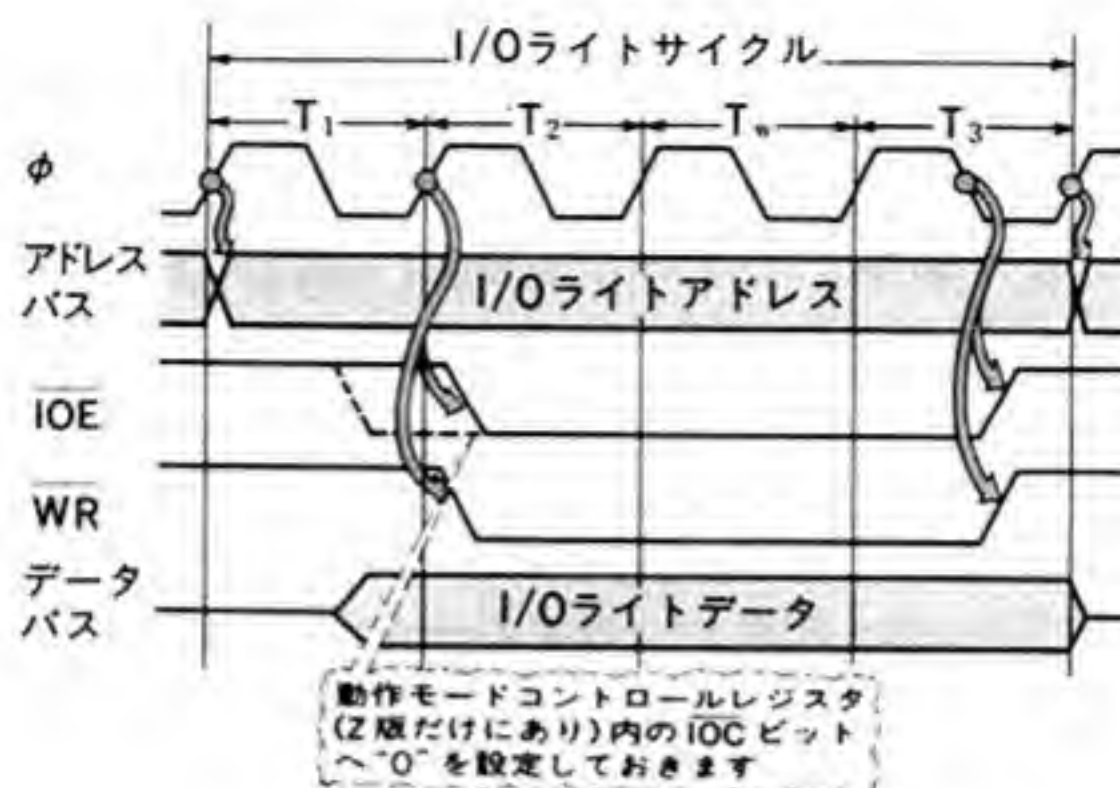


図 3・6 I/O ライトサイクル (IOC = "0" の場合)

64180Z 版

64180Z (Z180) では、Z80 周辺 LSI とのインタフェースが容易になります。64180R1 版から、 $\overline{\text{LIR}}$ 、 $\overline{\text{IOE}}$ 、 $\overline{\text{RD}}$ タイミングと RETI 命令の実行タイミングが改善されています。Z80 周辺 LSI を使用するシステムには、64180Z が適しているでしょう。

3・3 オペコードフェッチ サイクルと命令の実行

〔1〕 **オペコードフェッチサイクル** このサイクル（図3・7）は、メモリリードサイクルと同様に T_1 、 T_2 、 T_3 から構成されていますが、次の点で異なります。

- ① $\overline{\text{LIR}}$ 信号が“L”になります。
- ② リードデータは、半クロック速く確定しなければなりません。 T_3 ステートの ϕ クロックの立上りで、CPU はオペコードを取り込みます。

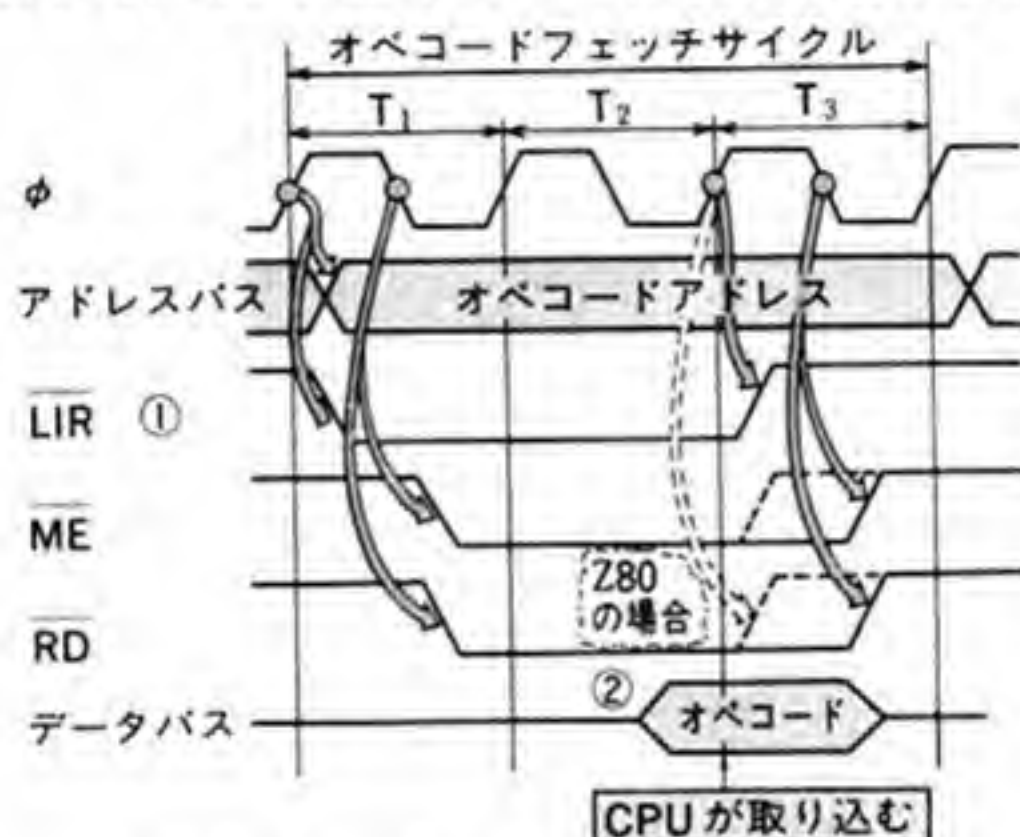


図 3・7 オペコードフェッチサイクル

また、オペコードフェッチサイクルでは、そのオペコードが1バイト目のオペコードであることを表すために、 $\overline{\text{ST}}$ 信号が“L”になります。タイミングは $\overline{\text{ME}}$ と同じです。

〔2〕 **Z80 との相違** Z80 では1サイクルが T_1 、 T_2 、 T_3 、 T_4 で構成されており、 T_3 と T_4 はリフレッシュサイクルとも呼ばれています。64180 では、リフレッシュサイクルはオペコードフェッチサイクルと独立しており、非同期に入ります。また、Z80 では $\overline{\text{MREQ}}$ と $\overline{\text{RD}}$ のインアクティブタイミングは T_3 の前半なので、64180 は半クロック遅くインアクティブとなります。

〔3〕 **命令の実行** CPU の基本動作は、1個または複数個のマシンサイクル (MC) によって構成されています。1マシンサイクルは、データのリード・ライトが伴う場合では T_1 、 T_2 、 T_3 の3ステートまたは T_1 、 T_2 、 T_w 、 T_3 の4ス

3. CPU タイミング

テートですが、データのリードライトが伴わない場合では、1マシンサイクルが1ステートです。なお、システムに使用しているメモリまたはI/Oのアクセスタイムが長い場合は、 T_3 の前に T_w を挿入してマシンサイクルを引き延ばします。

〔4〕 命令の実行タイミング例 I/O 命令 OUTO (m), A の例を示します。この命令は、2つのオペコードフェッチと1つのオペランドリード、さらにI/OライトとCPU内部状態から成ります。図3・8の例では、メモリ空間をアクセス時 T_w が挿入されない場合を示しています。

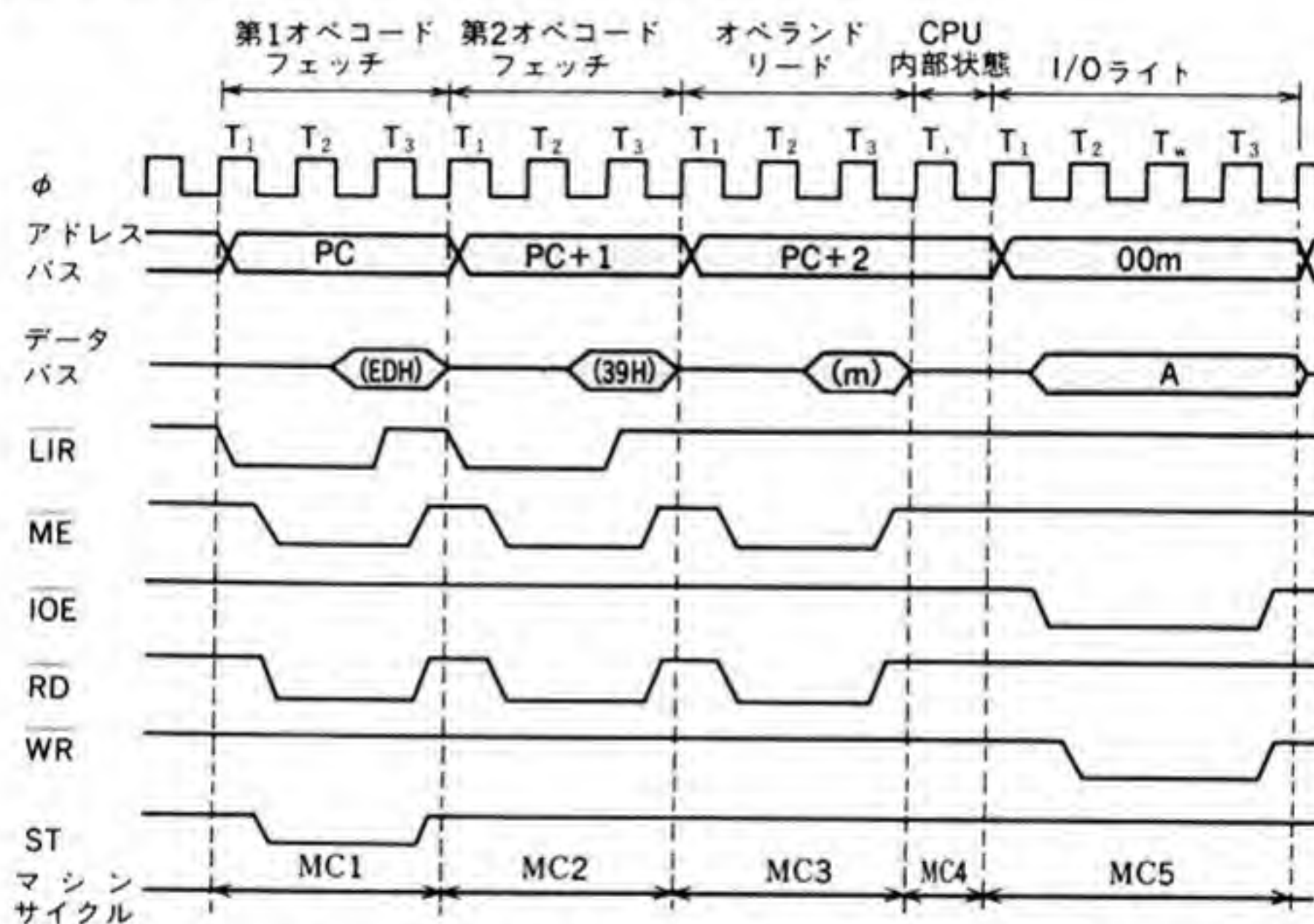


図 3・8 基本命令実行タイミング(OUTO(m), Aの場合)

4. メモリマネジ メントユニット

64180 は、8 ビットの CPU で世界で初めて 1 Mバイトのアドレス空間をサポートした MMU (メモリマネジメントユニット) を内蔵しています。従来の 8 ビット CPU はプログラムエリアが 64K バイトという制約があり、多くの応用機器においてアドレス空間が不足していました。64180 は、こうした大容量のメモリを必要とする応用機器で使用されることが多いバンクスイッチ方式の MMU を内蔵しています。

4・1 MMU の 概 念

〔1〕 64180 内蔵 MMU の機能 Z80をはじめ、8ビット CPUは16ビットのアドレスをもっています。したがって8ビット CPUは、接続できるメモリが $2^{16} \rightarrow 64\text{ K}$ バイトとなります。

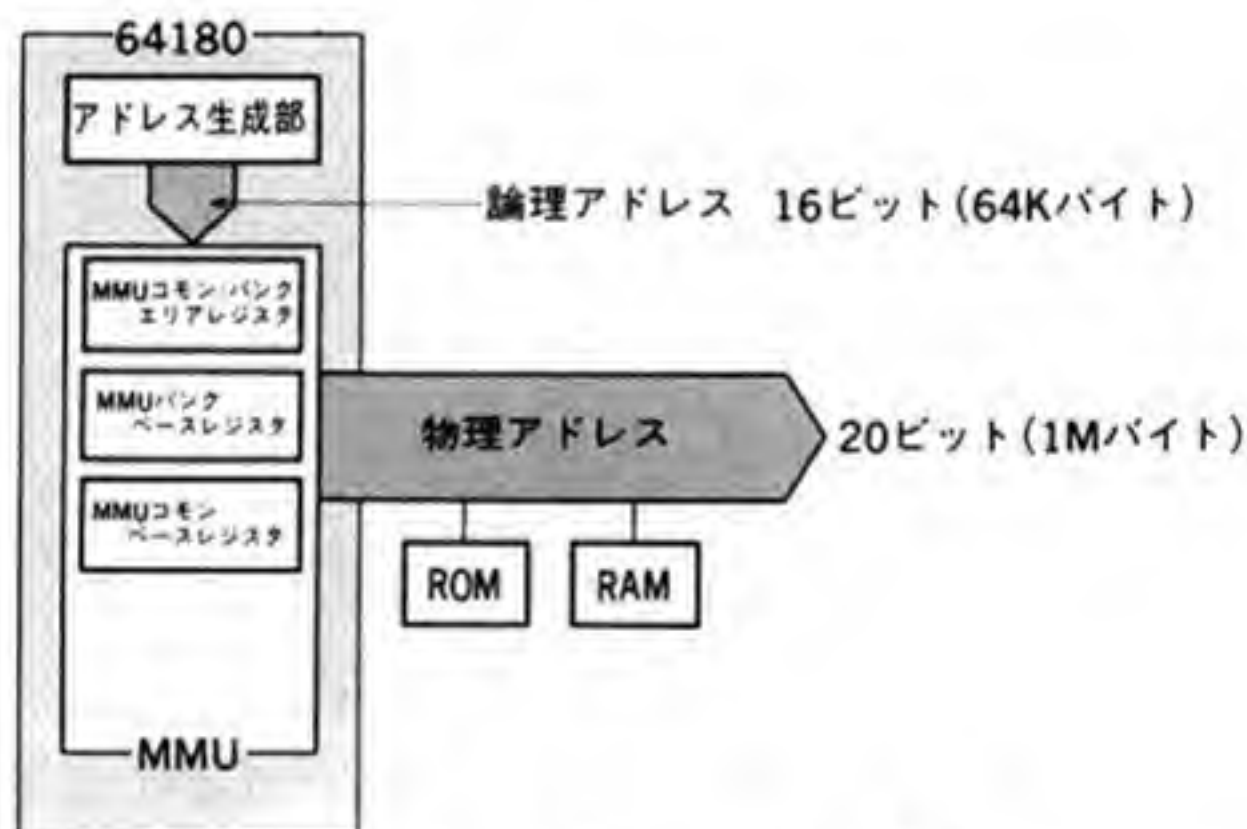
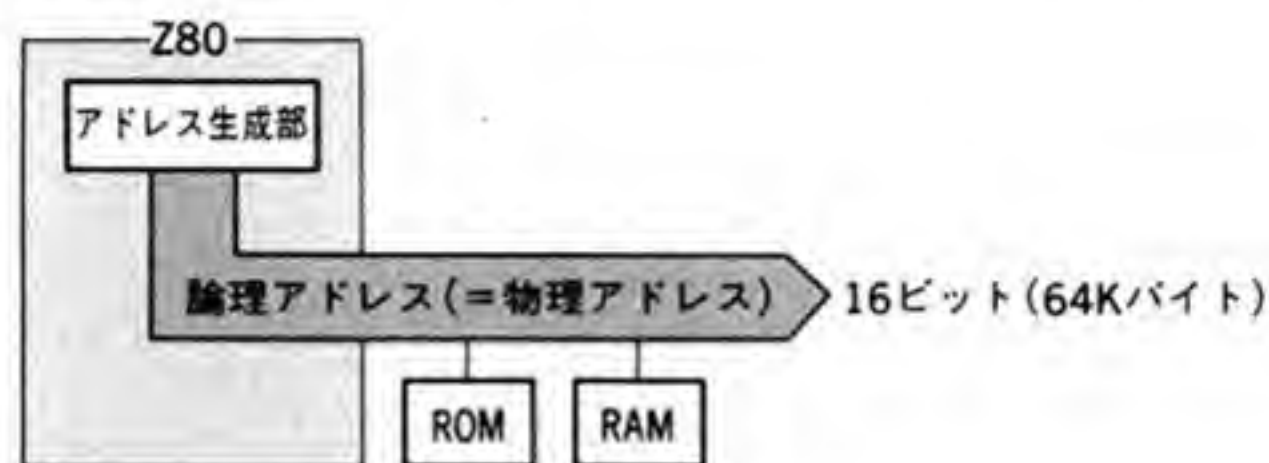


図 4・1 64180 の MMU の機能と構成

一方、現在多くの応用分野において、64 K バイトではメモリ空間が不足することが多くなっています。64180 のメモリマネジメントユニット (Memory Management Unit) は、バンクスイッチにより 64 K バイトを超える大容量のメモリを接続可能とします。具体的には、64180 の MMU は最大 20 ビットの物理アドレスを生成します。すなわち、64180 は $2^{20} \rightarrow 1\text{ M}$ バイトのメモリ空間をもつことができます (図 4・1)。

〔2〕 64180 内蔵 MMU の構成 64180 の MMU は、16 ビットの論理アドレ

スを 20 ビットの物理アドレスに変換するため、3つのレジスタをもっています。これらのレジスタは、MMU コモン/バンクエリアレジスタ、MMU バンクベースレジスタ、MMU コモンベースレジスタと呼ばれ、I/O 空間上に配置されています。64180 の MMU は、この 3 つのレジスタと制御、演算回路により構成されています。

〔3〕 **MMU の動作範囲** 64180 の MMU は、CPU がメモリをアクセスする場合に動作します。これは以下に示す 4 通りの場合です。

- (1) オペコードフェッチ
- (2) メモリへのデータのリード/ライト
- (3) 割込みベクタリングアドレス生成
- (4) 割込みのリスタートアドレス生成

同じメモリへのリード/ライトでも、内蔵の DMAC 動作中は MMU はバイパスされ、内蔵 DMAC から直接物理アドレスが出力されます。また、I/O 空間へのリード/ライトの場合も MMU はバイパスされます。この場合、 $A_{15} \sim A_0$ には有効な I/O アドレスが出力されますが、 $A_{19} \sim A_{16}$ には OOH が出力されます。

4・2 MMU の 動 作

4・2・1 エリアレジスタの機能

〔1〕 論理空間の分割 64180 の MMU は、64 K バイトの論理アドレス空間を最大 3 つのエリア（コモンエリア 0、バンクエリア、コモンエリア 1）に分割します。この分割の指定を行うのが、**MMU コモンバンクエリアレジスタ**（以下エリアレジスタと略す）です。

このエリアレジスタは 8 ビットから成り、上位 4 ビットでコモンエリア 1 の下限を、下位 4 ビットでバンクエリアの下限を指定します。エリアレジスタの構成と機能を図 4・2 に示します。これら 4 ビットの指定は、おのおのの空間の最上位 4 ビットのアドレス（A₁₅～A₁₂）に対し有効となります。

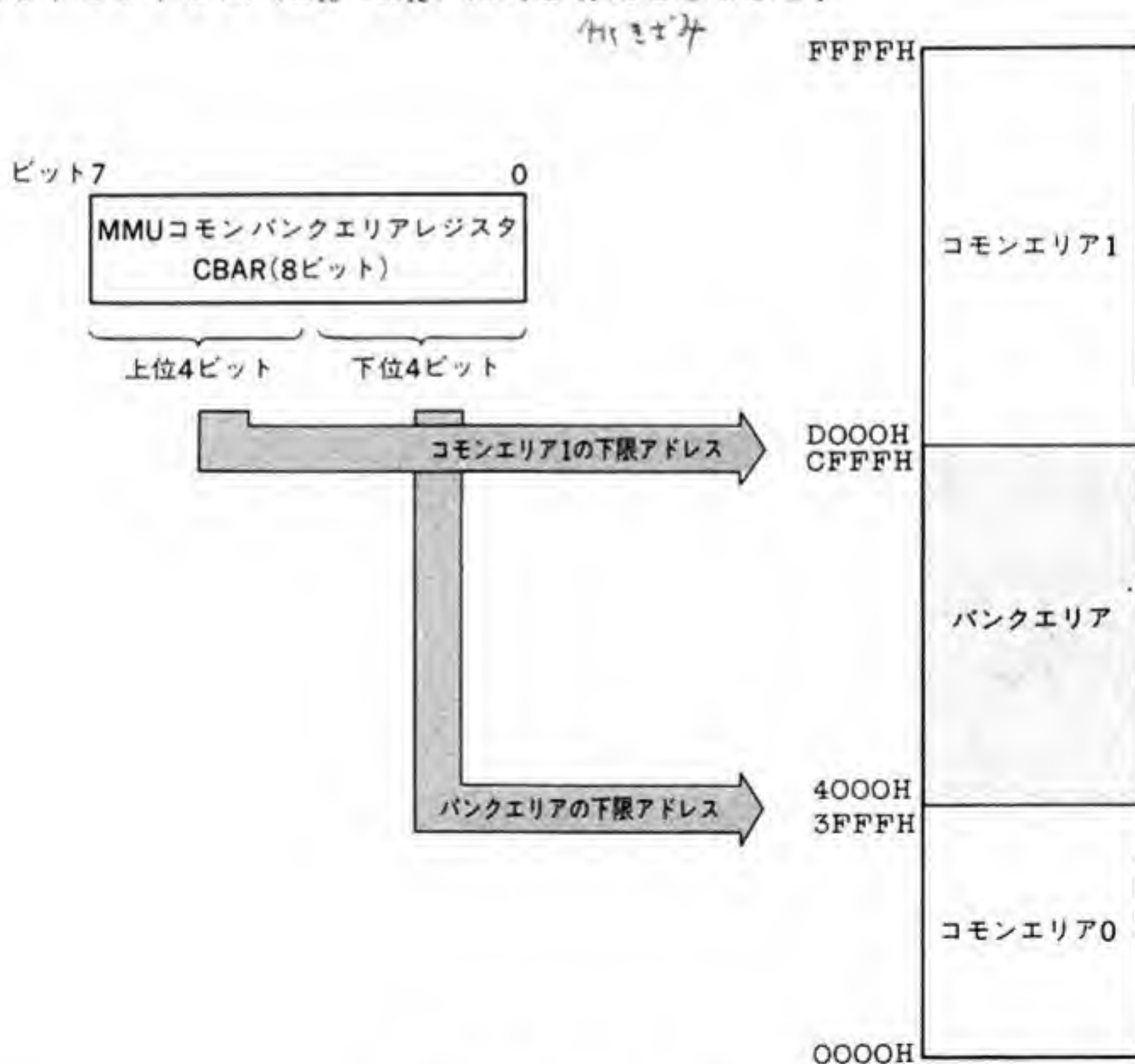


図 4・2 論理空間の分割

〔2〕 エリアレジスタの設定例 たとえば、エリアレジスタに 'D4H' の設定を行うと、コモンエリア1の下限の指定は 'D' となり、D000H から FFFFH がコモンエリア1となります。同様に、バンクエリアの下限の指定は '4' となり、4000H から CFFFH がバンクエリアとなります。コモンエリア0の下限は常に0であるため、0000H～3FFFH がコモンエリア0となります(図4・3)。

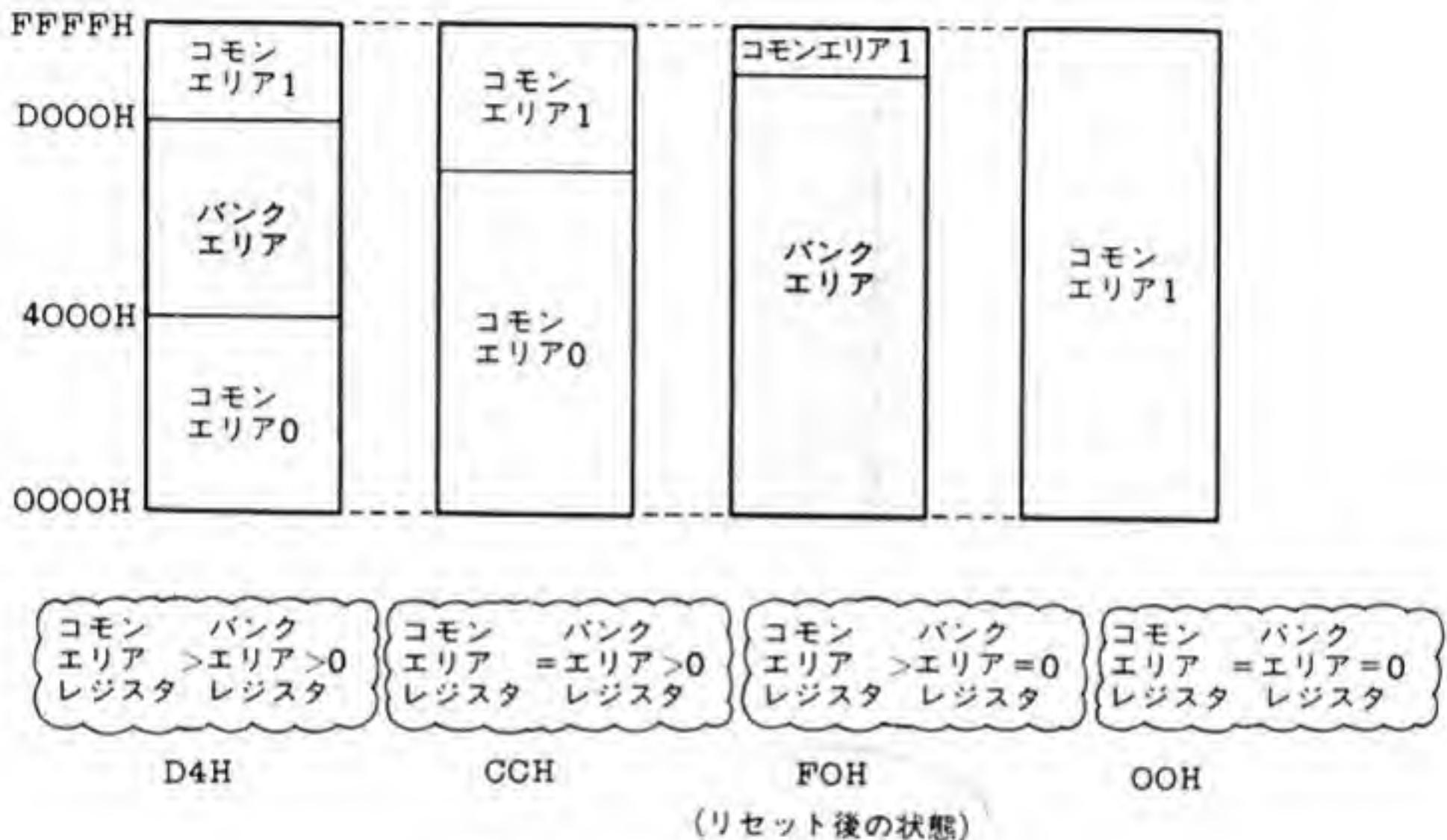


図 4・3 エリアレジスタの設定例

これ以外にも、コモンエリア1の下限=バンクエリアの下限(>0)の設定により、コモンエリア1とコモンエリア0により2分割の構成をとったり、コモンエリア1の下限>バンクエリアの下限=0の設定により、コモンエリア1とバンクエリアにより2分割の構成をとったり、コモンエリア1の下限=バンクエリアの下限=0の設定により、コモンエリア1により論理空間を構成することもできます。このように、エリアレジスタの設定により4通りの論理アドレス空間の組合せのうち1つを選び、各エリアの空間の大きさを指定することができます。

なお、エリアレジスタはリセット直後 FOH に初期化されます。

コモンエリア1の下限アドレスとバンクエリアの下限アドレスは、常にコモンエリア1の下限アドレスが大きくなるように設定する必要があります。すなわちバンクエリア \geq 0となる必要があります。

4. メモリマネジメントユニット

4・2・2 ベースレジスタの機能

[1] ベースレジスタ ベースレジスタ(MMUコモンベースレジスタ, MMU

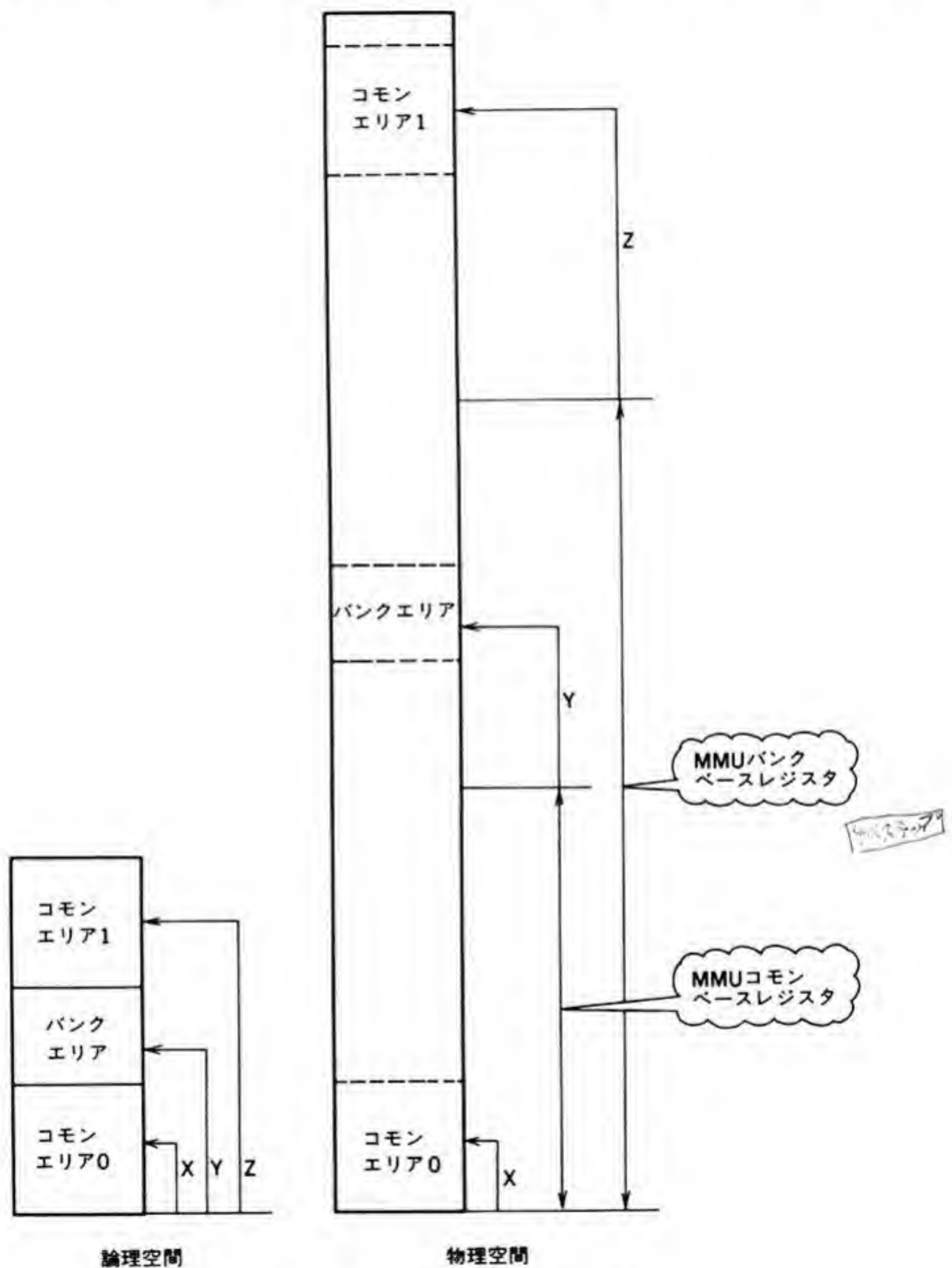


図 4・4 論理空間から物理空間への写像

バンクベースレジスタを総称してベースレジスタと呼び、以下この2つのレジスタをベースレジスタと略す)は、エリアレジスタによって分割された3つのエリアのうち、コモンエリア1とバンクエリアに対しオフセット値を与えるレジスタです。すなわち、この2つのエリアの論理アドレスに対してはベースレジスタの値が加算され、実際にアクセスされる物理アドレスが決定されます。それぞれコモンエリア1に対してはコモンベースレジスタ、バンクエリアに対してはバンクベースレジスタが対応しています(図4・4)。

〔2〕 物理アドレスの生成 たとえば、バンクエリア上のメモリ(論理アドレスで指定)をアクセスしたとします。このときの論理アドレスとバンクベースレジスタの値が加算され、物理アドレスが生成されます。実際の加算は次のように行われます。論理アドレスの最上位の4ビット($LA_{15} \sim LA_{12}$)とベースレジ

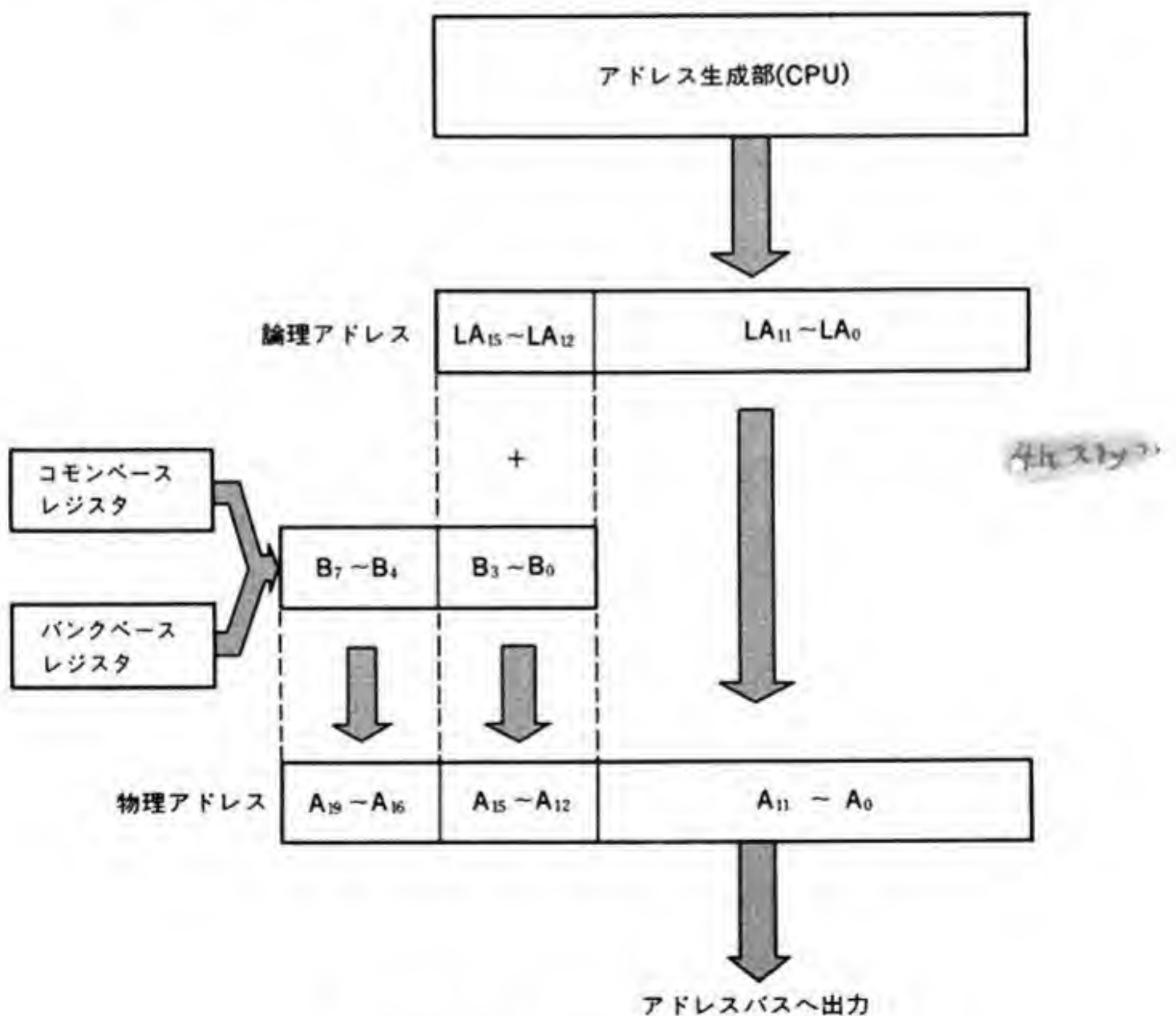


図 4・5 物理アドレスの生成

4. メモリマネジメントユニット

スタの下位4ビット ($B_3 \sim B_0$) が加算され、物理アドレスの $A_{15} \sim A_{12}$ が生成されます。また、この加算の際のキャリーとベースレジスタの上位4ビット ($B_7 \sim B_4$) が加算され、物理アドレスの $A_{19} \sim A_{16}$ に出力されます。論理アドレスの下位12ビット ($LA_{11} \sim LA_0$) はスルーで物理アドレスの $A_{11} \sim A_0$ となります。この物理アドレスの生成の様子を図4・5に示します。

これはコモンエリア1についても全く同様で、この場合のオフセット値としてはコモンベースレジスタが使用されます。ただし、コモンエリア0についてはオフセット値は常に '0' となります。すなわち、コモンエリア0は常に論理アドレス=物理アドレスであり、物理空間上の0番地から配置されます。

なおベースレジスタは、リセット直後コモンベースレジスタ、バンクベースレジスタとも OOH に初期化されます。したがって、この状態では論理空間=物理空間になっており、Z80 同様 64K バイトの論理アドレス空間をもつ CPU に見えます。

4・3 MMUの使用例

4・3・1 ROMと256KDRAMのアクセス

〔1〕 マッピングの検討 64180のMMUを使用し、32KバイトのEPROM(27256)と256KバイトのRAM(DRAM 日立HM51256など)をアクセスする方法を考えてみましょう(図4・6)。

プログラムエリアとして64Kバイトの論理アドレス空間のうち32KバイトをROMに割り当てます。これでプログラムは論理空間上に常駐することになります。残りの32KバイトをRAMエリアとして割り当てます。ここでスタックなどのワークエリアを設けるため、4Kバイトだけはコモンエリア1に割り当てます。コモンベースレジスタは固定とし、論理空間上の最上位に常駐させます。残った28Kバイトはバンクエリアに割り当て、バンク切替えを行ってデータメモリエリアとして使用します。

〔2〕 エリアレジスタの設定 前述のマッピングを実現するため、エリアレジスタにF8Hを設定します。この設定によりワークエリアのためのコモンエリア1がFOOOHからFFFFH、データメモリのためのバンクエリアが8000HからEFFFH、プログラムエリアのためのコモンエリア0が0000Hから7FFFHとなります。

〔3〕 ベースレジスタの設定 256KDRAMは、64180の1Mバイトのアドレス空間のC0000HからFFFFFFHに割りつけます。このうちFF000H～FFFFFFHの4KBをコモンエリア1に割り当て、ワークエリアとして使用します。コモンベースレジスタにはFOHを設定します。この値は固定とし書き換える必要はありません。DRAMの残りの252Kバイトを28Kバイトのバンクエリアのバンク切替えでアクセスします。この場合9個(252÷28)のバンクをもつことができます。その分割は、バンクエリア(1)はC0000H～C6FFFFH、バンクエリア(2)はC7000H～CDEFFFFHなどとなります。たとえば、バンクエリア(1)をアクセスしようとした場合、バンクベースレジスタにB8Hを設定することになります。

4. メモリマネジメントユニット

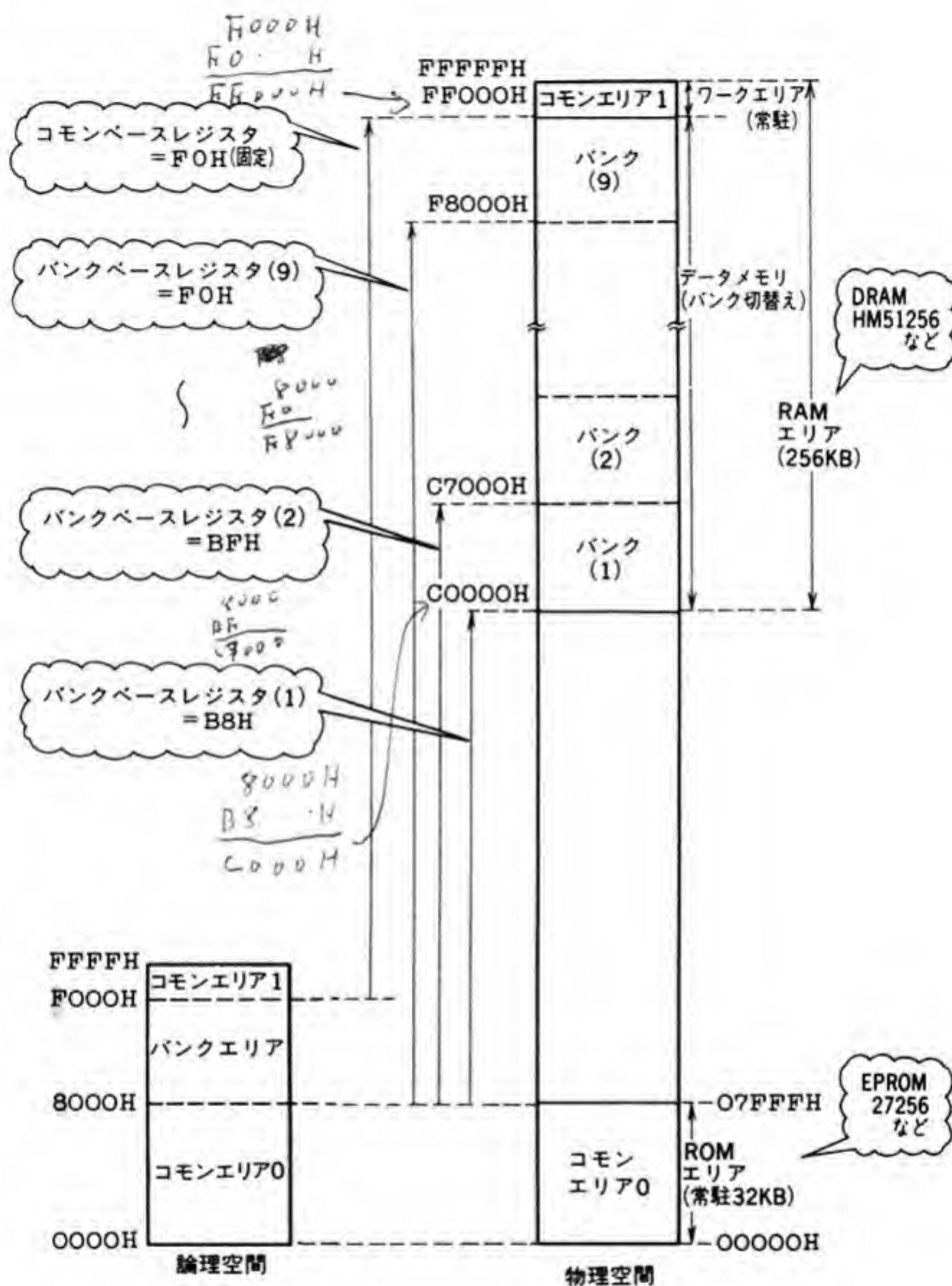


図 4・6 32Kバイト ROM と 256Kバイト RAM のアクセス

4・3・2 1Mバイトを超えたアドレスのアクセス

コモンエリア1によるコモンエリア0のアクセス 64180のMMUでは、論理アドレスとベースレジスタの和が20ビットのアドレス幅を超える場合があります。たとえば、コモンエリア1がFOOOH～FFFFHの場合、コモンベースレジスタにF1Hを設定したとします。この結果は、コモンエリア1に相当する物理アドレスが10E000H～10EFFFHとなり、20ビットの物理アドレス幅を超えています。この場合、オーバフローした21ビット目の値は無視され、実際の物理アドレスとしては0E000H～0EFFFHが出力されます。すなわち、FFFFHを超えた物理アドレスが生成されるとキャリーが無視され、00000Hからアクセスされることになります。

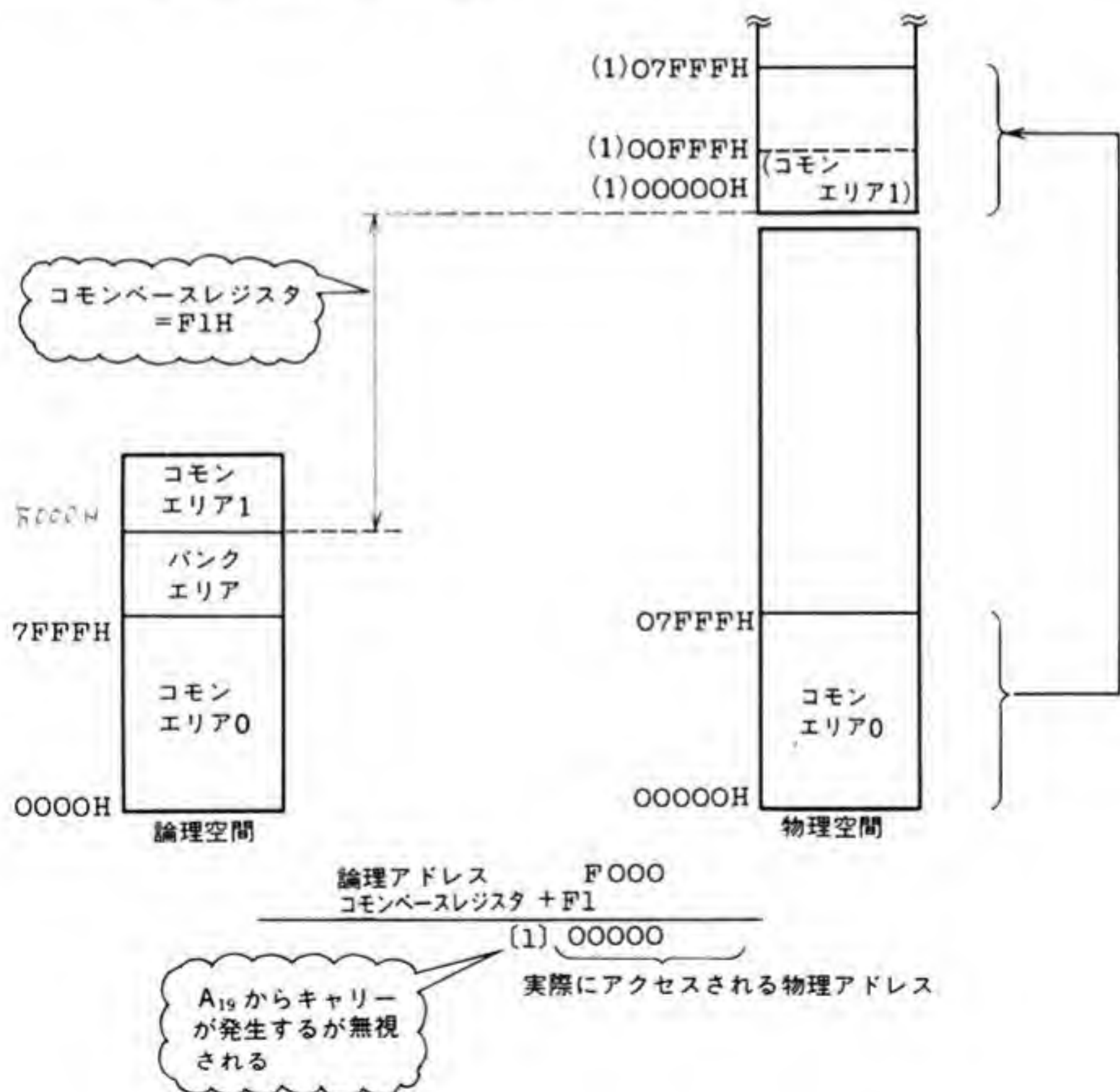


図 4・7 1Mバイトを超えたアドレスのアクセス

4. メモリマネジメントユニット

これを利用して、コモンエリア1のウィンドーからコモンエリア0に相当する論理アドレスをアクセスすることができます。たとえば、コモンベースレジスタに $F0H$ を設定することにより、コモンエリア0の $0000H \sim 0FFFH$ がアクセスされ、 $F0H$ を設定すると $1000H \sim 1FFFH$ がアクセスされます(図4・7)。

ベースレジスタ設定時に自分自身を制御するベースレジスタを変更すると、プログラムが複雑になるため、注意が必要です。

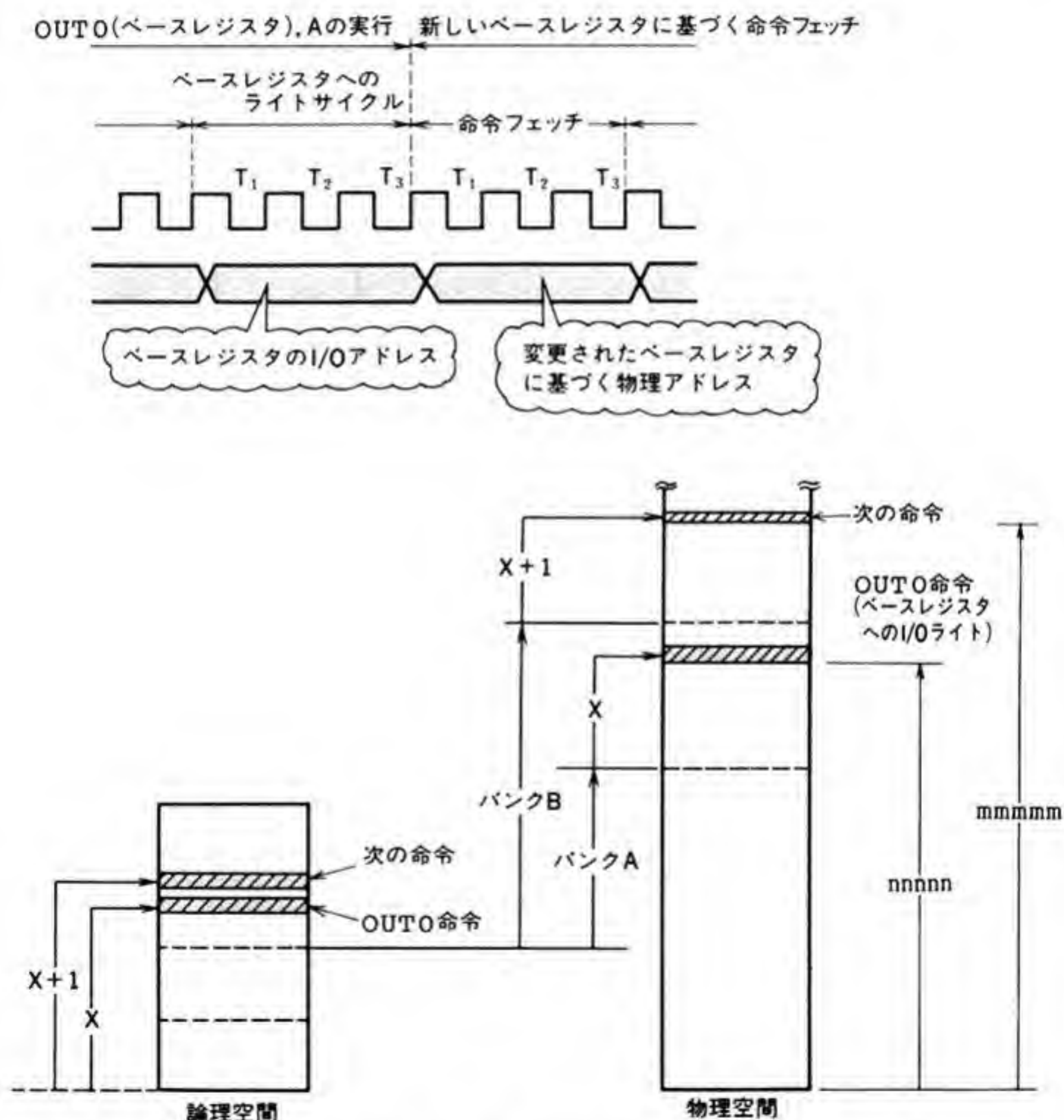


図4・8 ベースレジスタの設定

4・3 MMU の 使 用 例

たとえば、コモンベースレジスタをコモンエリア1から変更した場合です。ベースレジスタの変更を行うと、ベースレジスタへのI/Oライトサイクルの直後から、新しいベースレジスタの値に基づいた物理アドレスからオペコードをフェッチします。したがって、ベースレジスタへのライト命令 [OUTO(m), Aなど] がコモンエリア1（バンクA）の nnnnn 番地にあると、次はコモンエリア1（バンクB）の mmmmm 番地からフェッチが行われます。したがって、コモンエリア1からコモンベースレジスタを変更する場合、プログラムの配置に工夫が必要です（図4・8）。

こうした手間を避けるため、ベースレジスタを変更する場合、バンクベースレジスタの書換えはコモンエリア0またはコモンエリア1、コモンベースレジスタの書換えはコモンエリア0かバンクエリアから行うのが便利です。

第Ⅱ編

周辺機能

システムインテグレーションとは、多くの周辺機能を1チップ上に内蔵することであり、64180はタイマ、非同期方式シリアルI/O、クロック同期方式シリアルI/O、WAITコントローラ、DRAMリフレッシュコントローラ、DMAコントローラなど豊富な周辺機能をオンチップしており、究極の8ビットマイクロコンピュータと呼ばれています。特に、2チャンネルの非同期方式シリアルI/Oと2チャンネルのDMAコントローラは、多くのシステムのプリント基板サイズを小さくしました。

5. タイマ

NC やプリンタなどの多くのアプリケーションでは、タイムベースやパルス出力などを制御するタイマが必要になることがあります。64180では、16ビットデータ長のプログラマブルリロードタイマを2チャンネル内蔵しています。そのうち、チャンネル1にはタイマ出力端子をもっています。これらのタイマを使用することにより、ソフトウェアの負担が軽減できます。

5・1 タイマの構造

64180 にはプログラマブルリロードタイマが2チャンネルあります。図5・1の各ユニットは、次のような動作をします。

〔1〕 **タイマデータレジスタ** データ長が16ビットのレジスタです。タイマ動作の起動がかかると、 ϕ クロック20個に1回デクリメントします。そして、レジスタの内容が0000Hになると、リロードレジスタの内容がタイマデータレジスタにコピー（リロード）されます。CPUはタイマデータレジスタを8ビット単位で読書きでき、リードする場合、通常まず下位8ビットを読み、続いて上位8ビットを読みます。

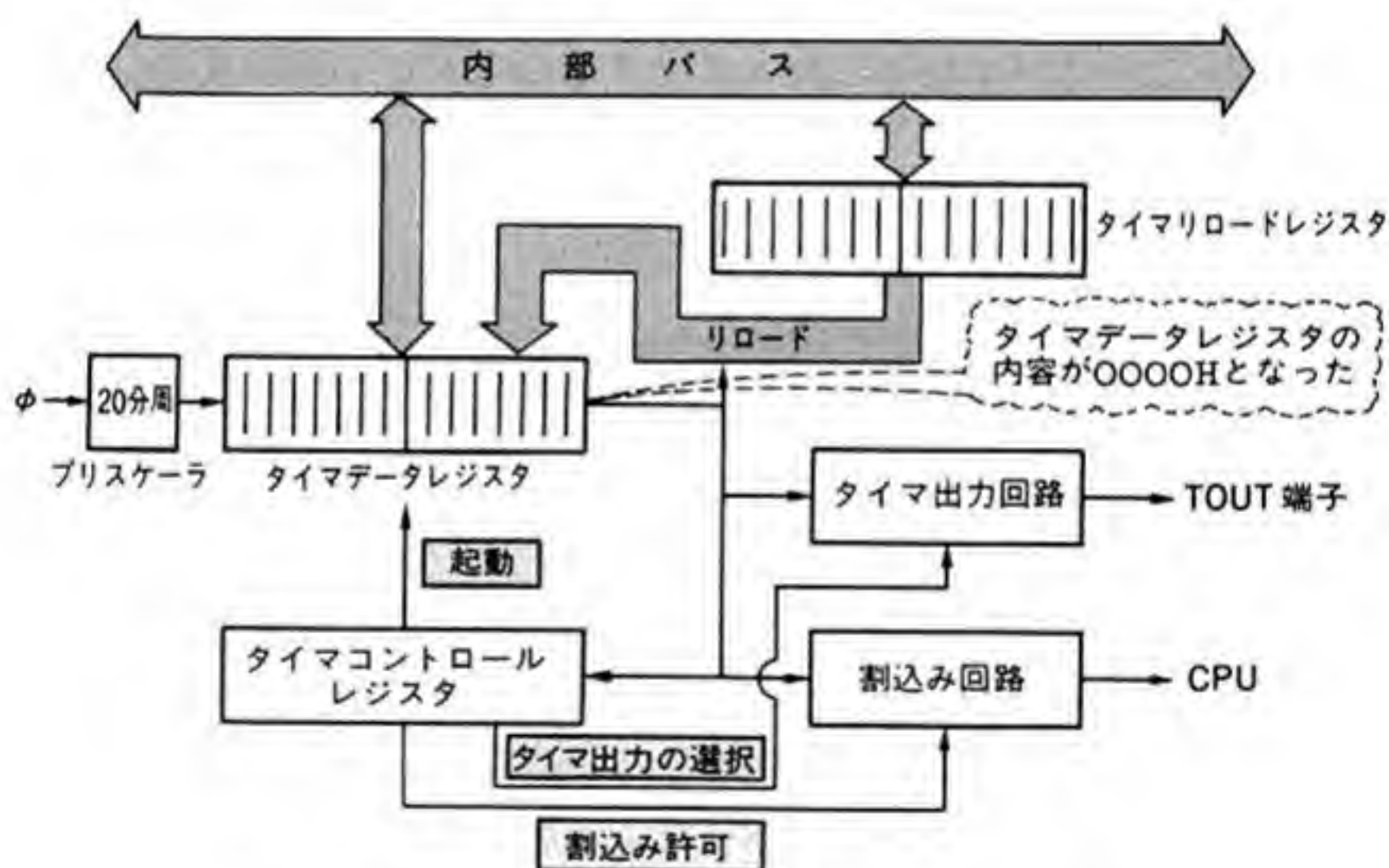


図 5・1 プログラマブルリロードタイマのブロック図

〔2〕 **タイマリロードレジスタ** データ長が16ビットのレジスタです。このレジスタは、タイマデータレジスタにコピーされる定数を保持しておくためのレジスタです。CPUからデータを8ビット単位で読書きできます。

〔3〕 **タイマ出力回路** TOUT 端子状態を制御する回路です。TOUT 端子は、アドレスの A₁₈ と兼用しています。“1”出力、“0”出力、トグル出力の3つのうちの1つを選択できます。なお、この回路はチャンネル0にはなく、チャネ

ル1だけにあります。

〔4〕 **割り込み回路** タイマデータレジスタの内容が0000Hとなると、タイマインタラプトフラグ (TIF) が“1”にセットされます。タイマインタラプトイネーブル (TIE) が“1”に設定されていると、CPUへ割り込みを要求します。

〔5〕 **タイマコントロールレジスタ** タイマ動作を制御するレジスタです。それぞれのビットの機能は図5・2に示されています。TOC0、TOC1ビットを除いて、“0”はチャネル0を“1”はチャネル1を表しています。

〔6〕 **プリスケアラ** 入力は ϕ クロックで、分周比は20です。しかしながらこの分周比は変更することができません。

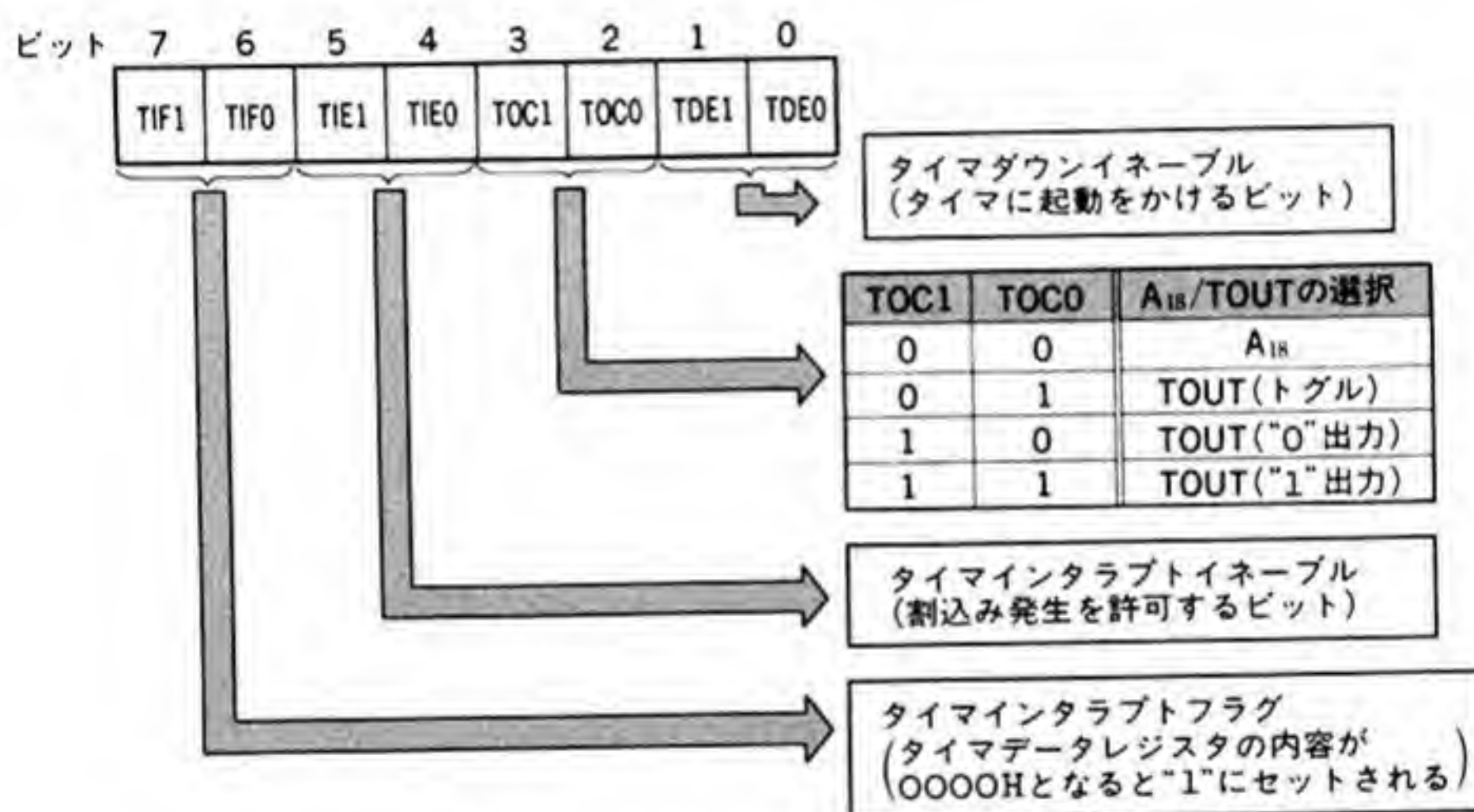


図 5・2 タイマコントロールレジスタ

タイマチャネル0の出力端子

タイマチャネル0には、タイマ出力端子がありません。しかし、タイマ出力が必要な場合には、ASCIの $\overline{\text{RTS}}_0$ 端子をタイマ出力端子とみなして使用できます。

$\overline{\text{RTS}}_0$ 端子の状態は、ASCIのコントロールレジスタ内にある $\overline{\text{RTS}}_0$ ビットの状態が反映されます。したがって、TOUT端子同様に使う場合、タイマ割り込みルーチン内のソフトウェアでこのビットに“1”または“0”を書き込めば実現できるでしょう。

〔2〕 **タイマ出力** トグル出力を選択している場合、図5・3のように前の状態を反転します。タイマリロードレジスタの値を変えなければ、デューティ比50%の矩形波が得られます。図では、途中でタイマリロードレジスタの値を変えているので、タイマ出力の周期が変化しています。

〔3〕 **タイマ割込み** アップカウント方式のリロードタイマと異なり、タイマリロードレジスタに値を設定した後の最初のリロード時からこの設定値が適用されます。この際、割込み周期は、タイマリロードレジスタの値に1を加えて 20 tcy を乗じた時間となります。ここで tcy は動作周波数の逆数です。TIFはタイマコントロールレジスタをリードした後、続けてタイマデータレジスタの上位または下位8ビットをリードすることによりリセットします。通常は、タイマ割込みルーチンの中でリセットします。

〔4〕 **タイマ出力の周期と割込み要求の周期** それぞれの最小値は、 80 tcy と 40 tcy です。そのときのタイマリロードレジスタの値は0001Hです。逆に最大値は、0000Hとした場合です。そのときのタイマ出力の周期は $65537(10000\text{H}+1)\times 40\text{ tcy}$ で、割込み要求の周期は $65537(10000\text{H}+1)\times 20\text{ tcy}$ です。

5.3 タイマの使用例

〔1〕音出力 タイマチャネル1を使用し、そのタイマ出力端子 TOUT よりパルスを出力して、音階を出す場合の例を示します。

TOUT 出力をトグル出力にした場合、タイマデータレジスタの内容が0000H となるごとに前の状態を反転するので、その反転する時間を調節することによりある一定周期の矩形波を出力します。そして、この矩形波をアンプを通じてスピーカに入力すれば、音として取り出すことができます（図5・4 参照）。

今、タイマリロードレジスタ1（RLDR1）の値を途中で変えなければ、その音の周期は図5・5の式で表されます。この式から、動作周波数とRLDR1の値を与えると、任意の周期の音が出力されます。逆に、希望する音を出すためには、RLDR1の値をこの式から逆算して求めればよいでしょう。

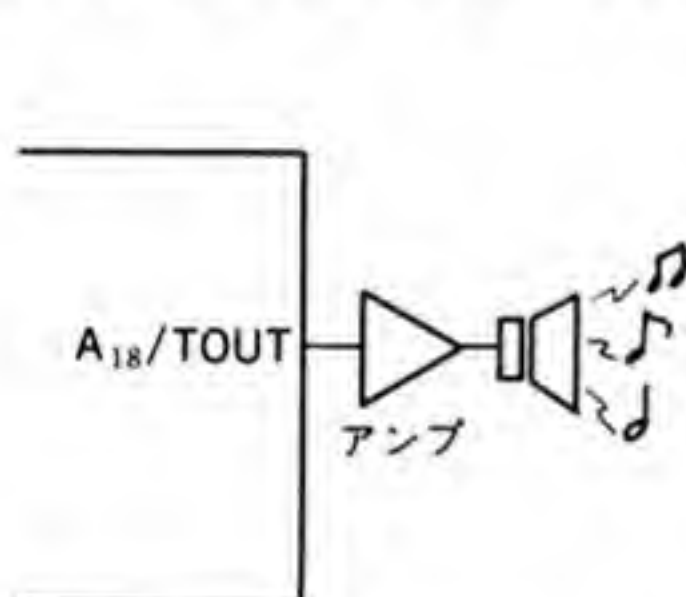


図 5・4 音出力

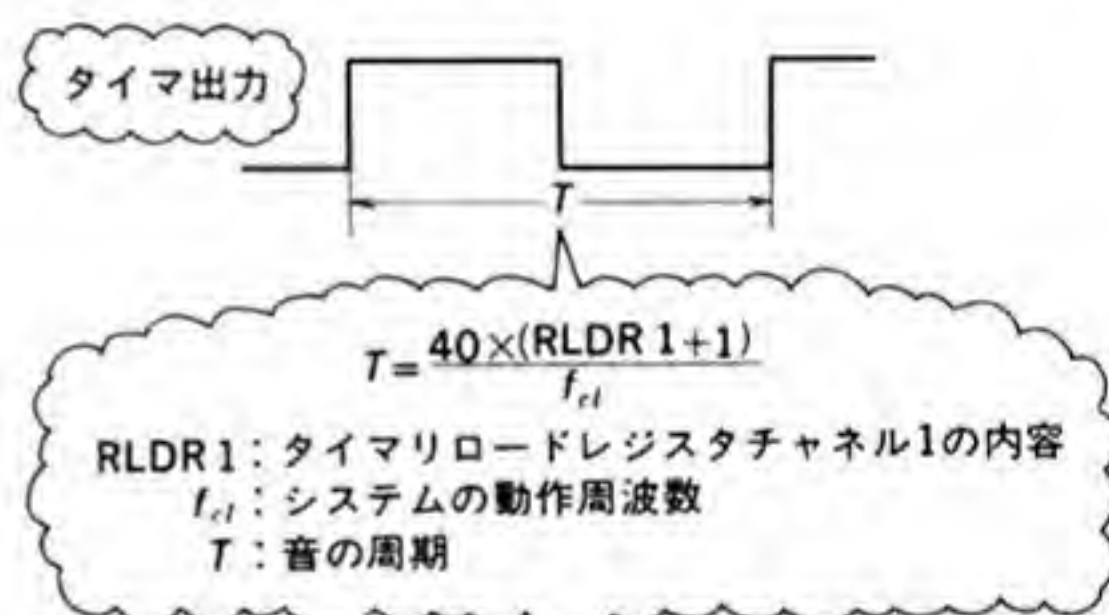


図 5・5 音の周期

表 5・1 音出力データ

音 階		RLDR1 の 値	
記 号	周波数 [Hz]	10 進	16 進
ド	523.25	293	0125
シ	493.88	310	0136
ラ	440.00	348	015C
ソ	392.00	391	0187
ファ	349.23	439	01B7
ミ	329.63	465	01D1
レ	293.66	522	020A
ド	261.63	586	024A

$f_{cl} = 6.144\text{MHz}$ 時

5・3 タイマの使用例

この方法により、7音階を与える RLDR1 の値を求めたものが表5・1です。ただし、動作周波数 f_{cl} は 6.144 MHz です。例として、「ラ」の音を出す場合のプログラミング例を図5・6に示しました。このリロードタイマを使用すれば、簡単なプログラムで任意の音を出せることがわかんと思います。

LD A, 5CH	
OUTO (TMDR1L), A	
OUTO (RLDR1L), A	タイマデータレジスタとリロードレジスタに「ラ」の音データを与える
LD A, 01H	
OUTO (TMDR1H), A	
OUTO (RLDR1H), A	
LD A, 00000110B	タイマ出力をトグルに選択し、タイマチャンネル1のイネーブルビットを
OUTO (TCR), A	
	"1"に設定して、タイマチャンネル1を起動させる

図 5・6 「ラ」の音を出すプログラム例

〔2〕 **タイムベース** 周期0.1秒のソフトタイマを考えます。割込みを使用し、CPUへ0.1秒ごとに割込み要求を出す場合です。割込み要求はタイマデータレジスタが 0000H となったとき発生するので、その周期は $T/2$ で与えられます。したがって、0.1秒ごとに割込み要求を発生させるために、タイマリロードレジスタへは $40\,000 - 1 = 39\,999$ (16進では、9C3FH) を与えます。ただし、図5・5の式を使用し $T = 0.2$ 秒、 $f_{cl} = 8\text{MHz}$ としています。この値を RLDR に書き込み、割込みをイネーブルにした後タイマを起動させます。

6. 非同期方式 シリアル I/O

非同期方式シリアル I/O は、パソコン、プリンタなど多くのアプリケーションで使用されている通信の代表的な方式です。ACIA や UART などいろいろな呼び方がありますが、64180 では ASCII と呼んでいます。ASCII は、スタートビットとストップビットをデータストリングの中にもっているのが特徴であり、スタートビットで同期をとるためクロック信号を送る必要がありません。

6・1 シリアルコミュニケーション インタフェースの分類

マイクロコンピュータに内蔵されているシリアルコミュニケーションインタフェースは、図 6・1 に示すように非同期方式シリアル I/O と同期方式シリアル I/O に分類することができます。

〔1〕 **非同期方式シリアル I/O** 別名「調歩同期式シリアル I/O」と呼ばれる非同期式シリアル I/O は、パソコン、プリンタなど多くのアプリケーションで使用されています。64180 ファミリーでは、この非同期方式シリアル I/O を **ASCI** (Asynchronous Serial Communication Interface) と呼んでいます。

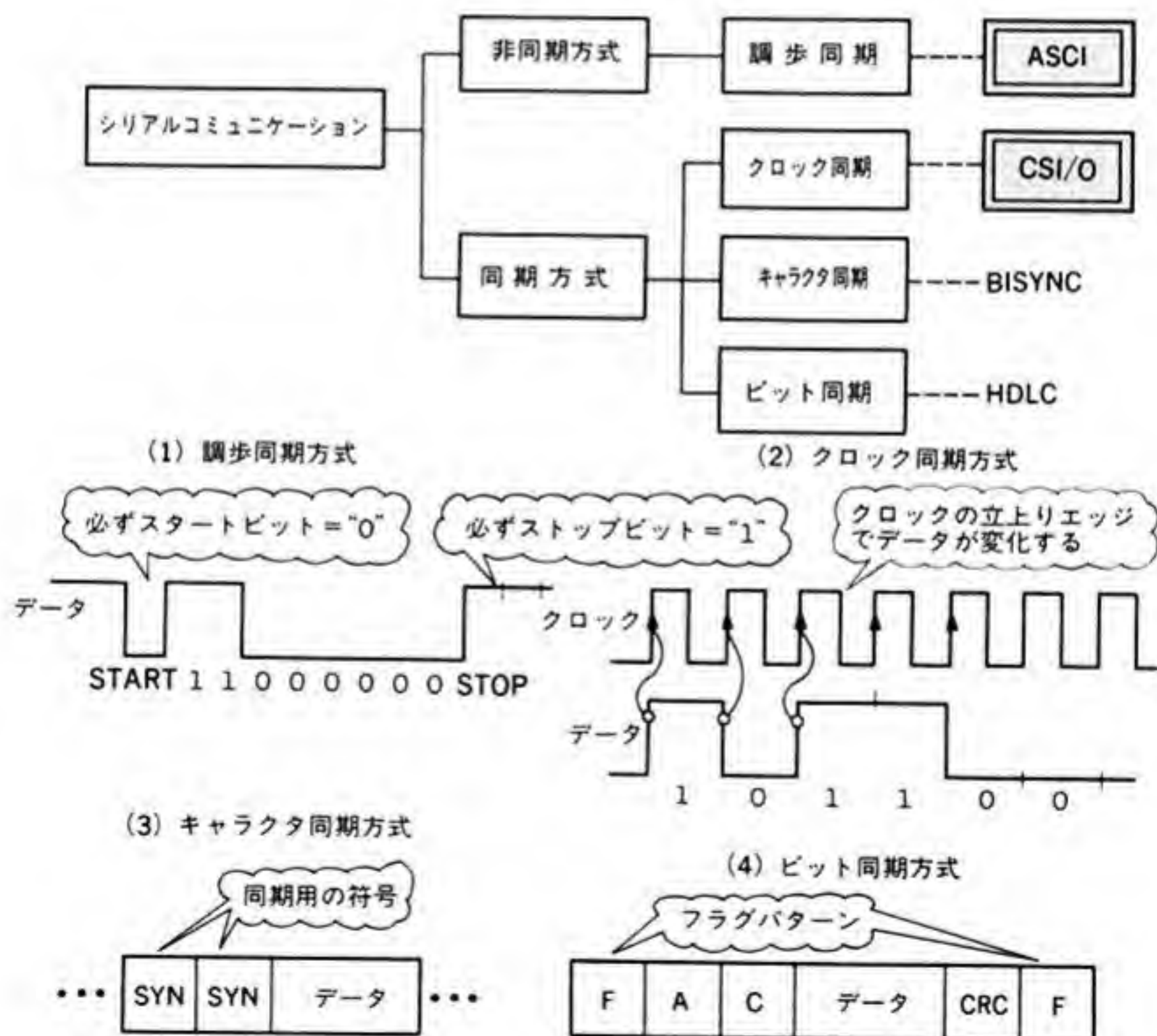


図 6・1 シリアルコミュニケーションインタフェースの分類

6・1 シリアルコミュニケーションインタフェースの分類

ASCIIの大きな特徴は、送信側のクロックタイミングと受信側のクロックタイミングが非同期でデータの送受信ができることです。このため、データストリングの中に同期をとるためのスタートビットおよびストップビットが必要です。

〔2〕 **同期方式シリアル I/O** 同期方式シリアル I/O は、①クロック同期方式、②キャラクタ同期方式、③ビット同期方式に分類できます。このうち 64180 ファミリーでは、クロック同期方式シリアル I/O (**CSI/O** : Clocked Serial Input Output) を内蔵しています。

CSI/O は、マイコンとマイコンのインタフェース、マイコンと周辺 LSI とのインタフェースなどに使用されます。CSI/O の特徴は送受信のビットごとの同期を、クロックの立上りエッジまたは立下りエッジでとることです。このため、送信側と受信側が同じタイミングのクロックを用いてデータのやりとりをすることが必要となります。

キャラクタ同期方式シリアル I/O は、同期をとるための特定符号 (SYN 符号) をデータの前につけて伝送する方式です。SYN 符号を 2 つつけてデータを送る方式を **BISYNC** と呼びます。SYN 符号を受信すると、その次からデータが連続して送られてくるものとして文字を組み立てていきます。

ビット同期方式シリアル I/O は、伝送路に一定のビットパターン (フラグパターン) を送出しておくことにより、送信側と受信側の同期をとる方式です。フラグパターン以外のデータを受信するとデータが送られてきたと判断し、再びフラグパターンを検出するまでその間の信号をデータとみなします。

6・2 ASCI の構成と機能

64180 ファミリーは、ASCI を 2 チャンネル内蔵しています。ASCI は図 6・2 に示すように、受信側がレシーブシフトレジスタ、レシーブデータレジスタから成り、送信側がトランスミットシフトレジスタ、トランスミットデータレジスタから成っています。さらに、送信/受信の制御のためのコントロールレジスタ 2 個と、ASCI の状態を示すステータスレジスタから構成されています。

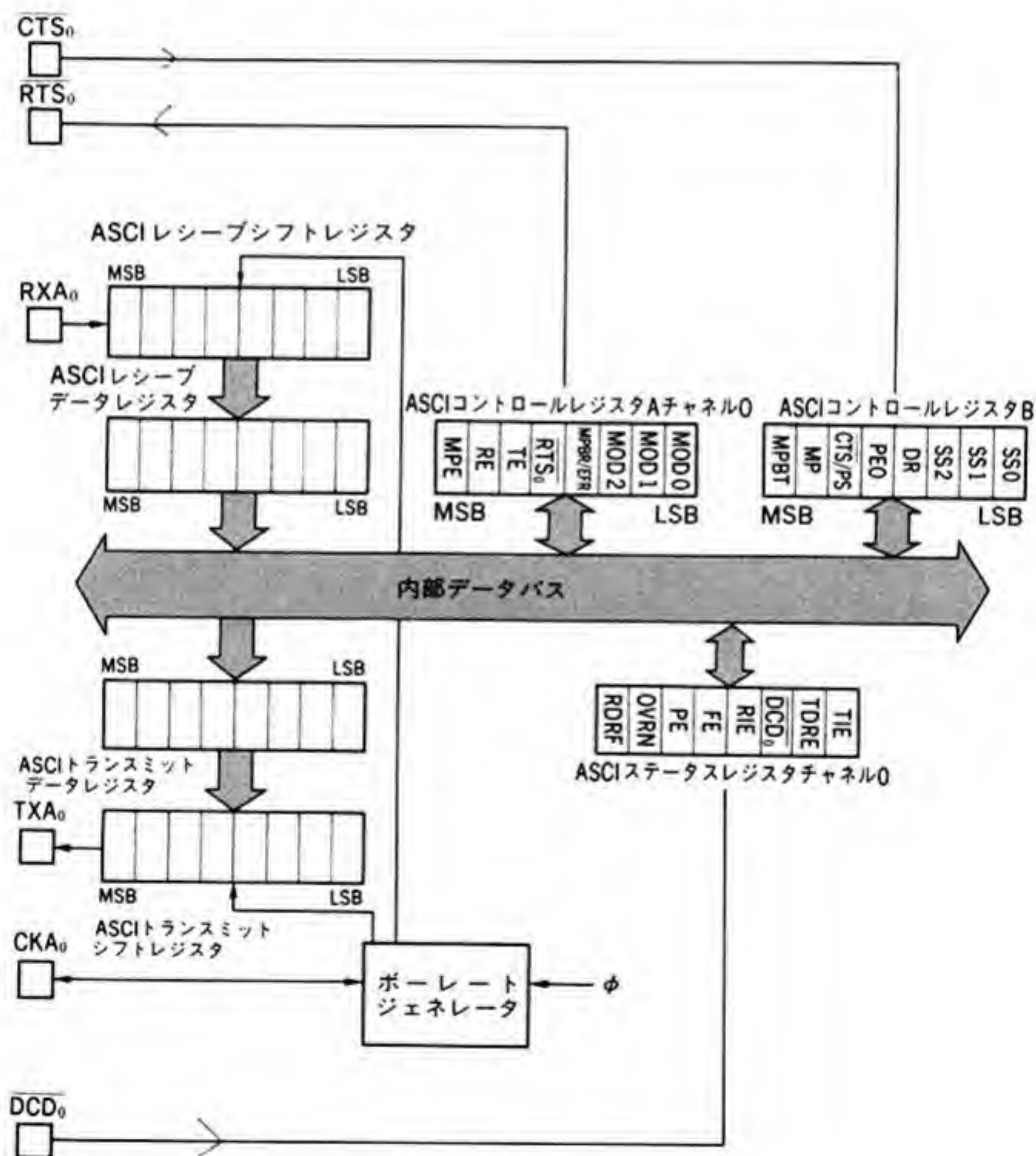


図 6・2 ASCI ブロック図

〔1〕 **ASCIIによるデータ受信** コントロールレジスタにある **RE** (Receive Enable) = “1” で受信可能な状態となります。RXA₀ 端子から入力されたデータはまず、レシーブシフトレジスタへ1ビットずつ取り込まれシフトされます。1バイトのデータの受信が終了すると、受信したデータは自動的にレシーブデータレジスタへ転送されます。この状態で次の新しい受信データが、再びレシーブシフトレジスタへ取込み可能となります。つまり、受信部はレシーブシフトレジスタとレシーブデータレジスタの2重バッファ構成となっていますので、1バイト受信後、次のデータの受信が完了する前までに受信データをレシーブデータレジスタから読めばよいようになっています。レシーブデータレジスタに受信データが送られてくると、ステータスレジスタの **RDRF** (Receive Data Register Full) が“1”にセットされます。

〔2〕 **ASCIIによるデータ送信** コントロールレジスタの **TE** (Transmit Enable) = “1” で送信可能な状態となります。1バイトの送信データは、まずトランスミットデータレジスタへ書き込まれます。トランスミットシフトレジスタがからになると、トランスミットデータレジスタから自動的にトランスミットシフトレジスタへ転送され、1ビットずつ TXA₀ 端子からデータが送信されます。送信側もトランスミットデータレジスタとトランスミットシフトレジスタの2重バッファ構成になっていますので、トランスミットデータレジスタがからになる(ステータスレジスタにある **TDRE**: Transmit Data Register Empty が“1”)と次の送信データを書き込むことができます。つまり、データをTXA₀ 端子から送信しながら次のデータも書くという並列処理ができますので、複数バイトのデータを高速に転送できます。

6・3 ASCII のデータフォーマット および転送速度

〔1〕 データフォーマット 図 6・3 に示すように ASCII コントロールレジスタの MOD0～MOD2 を用いることにより、送受信のデータフォーマットを選択できます。データフォーマットは ① スタートビット、② データ、③ パリティビット、④ ストップビットから構成されます。スタートビットはデータの最初のビット位置を示し必ず“0”です。一方、ストップビットはデータの最後を示し必ず“1”です。ストップビットは 1 ビットまたは 2 ビットのいずれかを選択できます。スタートビットに続くデータは 7 ビットと 8 ビットがあり、用途によりユーザが使い分けることができます。たとえば、ASCII コードを送受信するときは 7 ビットのデータを使用します。

MOD2	MOD1	MOD0				
0	0	0	START	7bit data	STOP	
0	0	1	START	7bit data	2STOP	
0	1	0	START	7bit data	PARITY	STOP
0	1	1	START	7bit data	PARITY	2STOP
1	0	0	START	8bit data	STOP	
1	0	1	START	8bit data	2STOP	
1	1	0	START	8bit data	PARITY	STOP
1	1	1	START	8bit data	PARITY	2STOP

図 6・3 データフォーマットの種類

6・3 ASCII のデータフォーマットおよび転送速度

さらに、データ送受信の際、伝送路におけるエラーを検出するためにパリティビットの付加もできます。偶数パリティか奇数パリティかをコントロールレジスタの **PEO** (Parity Even or Odd) ビットで選択できます。このように、8種類のデータフォーマットが選択できます。

〔2〕 転 送 速 度 転送速度(一般にボーレート)を発生する方法は、図 6・4 に示すように内部クロックを使用する場合と外部クロックを使用する場合の 2通りがあります。

内部クロックからボーレートを発生する場合、内部クロック ϕ はボーレート選択器、プリスケアラおよびサンプリングレート選択器を通して任意のボーレートとなります。

一方、外部クロックからボーレートを発生する場合、外部クロック端子 CKA より入力されたクロックは、プリスケアラとボーレート選択器を通らず直接サンプリングレート選択器の入力となり、ボーレートとなります。

サンプリングレート選択器は、ボーレートの 16 倍または 64 倍のサンプリングクロックを生成するために必要であり、この指定はコントロールレジスタ B の DR (Divide Ratio) ビットで行います。また、内部クロックからボーレートを生成する場合に限り、プリスケアラ (÷10 または ÷30 の選択) およびボーレート選択器 (÷1 ~ ÷64) が有効となります。

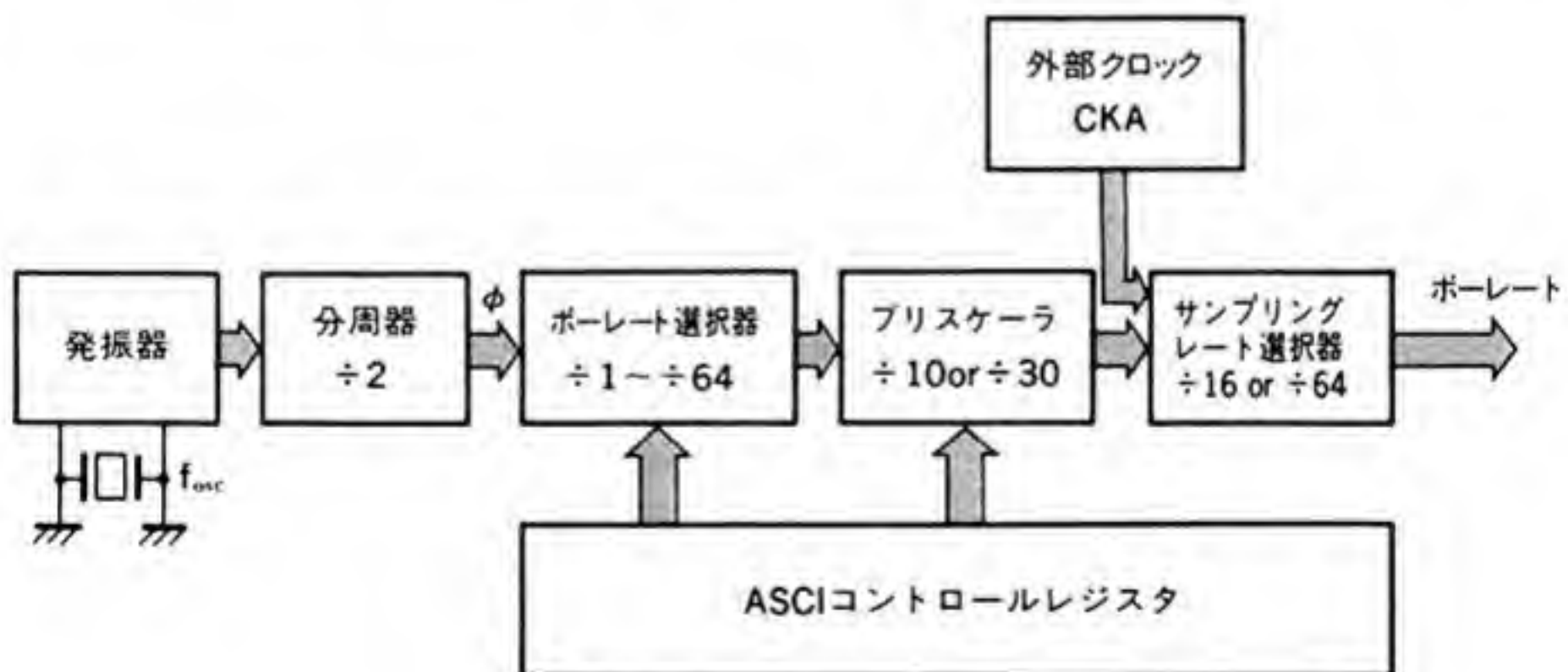


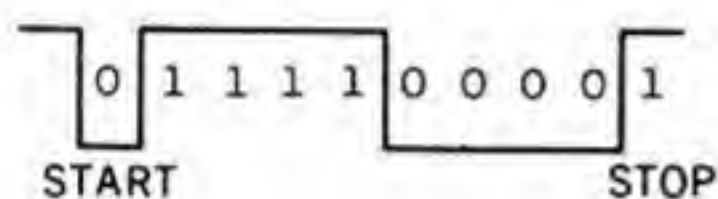
図 6・4 ASCII のクロック発生回路

6・4 ASCII のエラーチェック機能

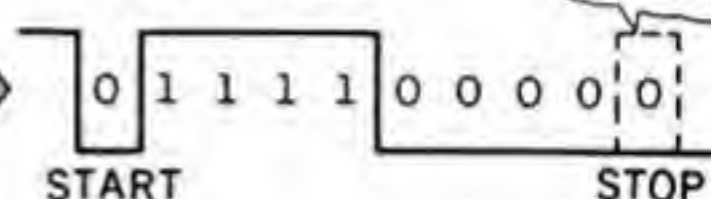
ASCII が備えているエラーチェック機能には、①フレミングエラー、②パリティエラー、③オーバランエラーの 3 種類があります。図 6・5 にエラーの種類と概要を示します。

〔1〕 フレミングエラー ASCII のデータフォーマットでは、データの最後

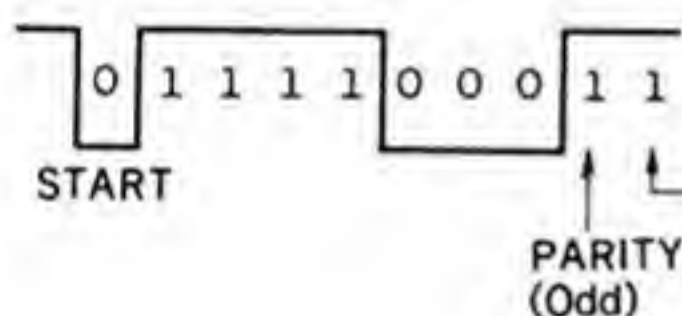
(1) フレミングエラー
〈正常〉



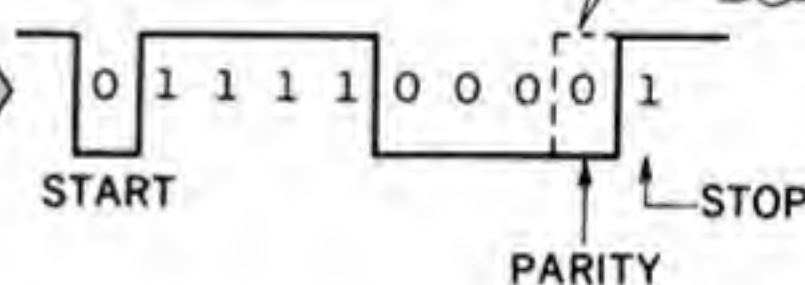
〈エラー発生〉



(2) パリティエラー
〈正常〉



〈エラー発生〉



(3) オーバラン

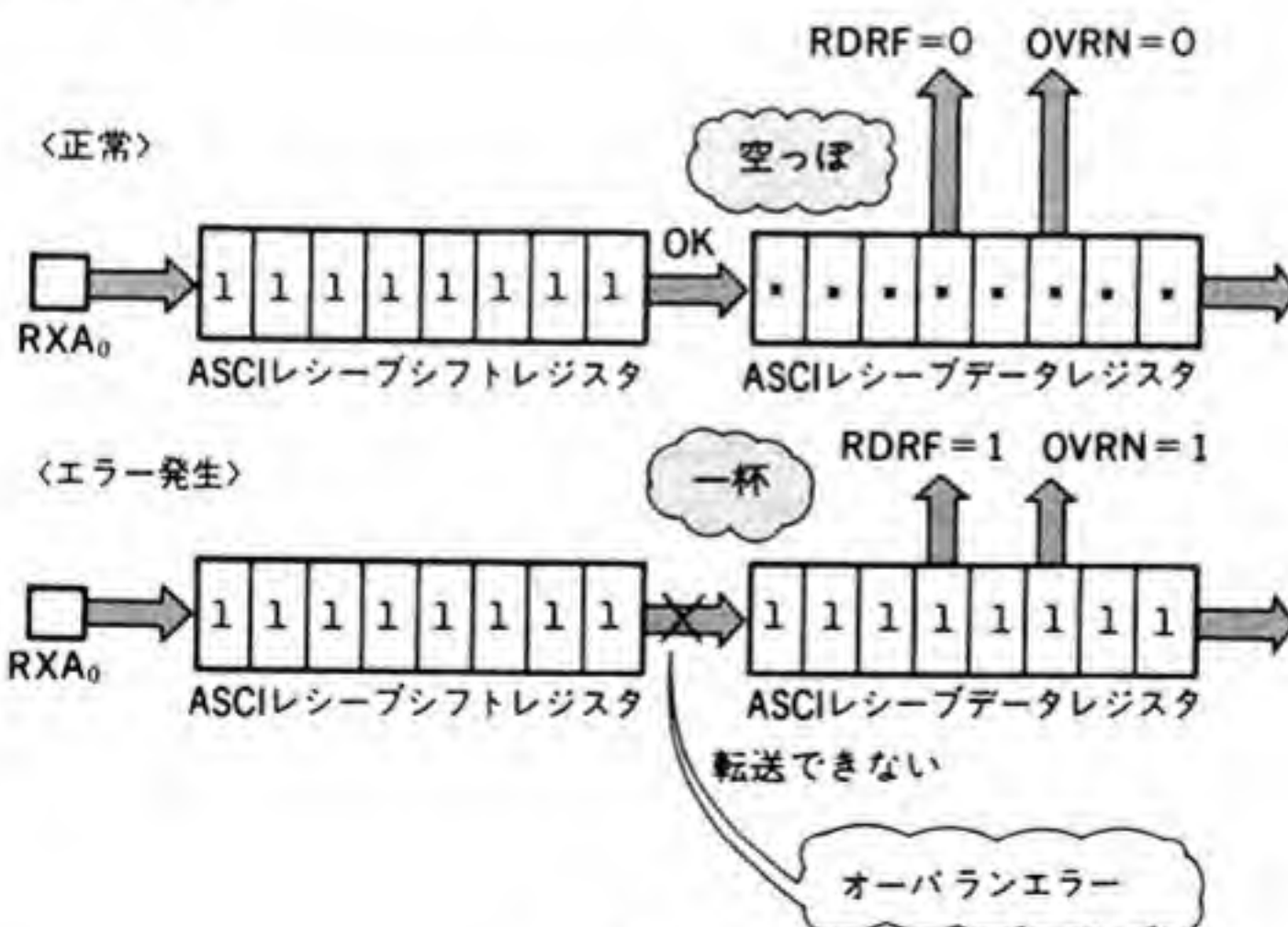


図 6・5 エラーの種類

は必ずストップビット（“1”レベル）でなければなりません。しかし、伝送路を通る間にストップビットが“1”から“0”に化けることがあります。このように変化したデータを受信すると、フレミングエラーが生じたとして、ASCIステータスレジスタのFE（Framing Error）フラグを“1”にセットします。

〔2〕 **パリティエラー** 送信データが伝送路を通り正しく受信されたかチェックするために、パリティエラーチェック機能を用います。データの最後にパリティビットをつけて、データとパリティビットの“1”の数が偶数になるようにするのを偶数パリティ、“1”の数が奇数になるようにするのを奇数パリティと呼びます。送信側と受信側であらかじめ偶数パリティか奇数パリティのどちらで交信するのかを決めておくと、受信データとパリティビットの“1”の数でエラー検出ができます。パリティエラーを検出するとASCIステータスレジスタのPE（Parity Error）フラグに“1”をセットします。

〔3〕 **オーバランエラー** ASCIの受信側は、ASCIレシーブシフトレジスタとASCIレシーブデータレジスタの2重バッファ構成ですが、受信したデータがASCIレシーブデータレジスタにまだ入っていて、CPUからまだ読まれていないときに次のデータの受信が完了した場合に、オーバランエラーが発生します。RDRF = “1”の状態で次の受信データの最後のビットまで受信したとき、ASCIステータスレジスタのOVRN（OVer RuN error）フラグを“1”にセットします。

これら4種類のエラーが発生すると、エラーフラグをセットするだけでなく割込みも発生させることができます。CPUはエラーチェックの結果に基づいて、それぞれのエラー処理を行い正しい送受信データのみ受け取ることが可能です。

6・5 モデム制御信号

モデム制御信号として、チャンネル 0 には \overline{CTS}_0 (クリア・ツー・センド), \overline{RTS}_0 (リクエスト・ツー・センド), \overline{DCD}_0 (データ・キャリア・ディテクト) の 3 端子があり、チャンネル 1 には \overline{CTS}_1 端子があります。これらのモデム制御信号は図 6・6 に示すように、モデムと接続することにより多くのシリアル通信を必要とするアプリケーションに使用できます。 \overline{CTS}_0 および \overline{RTS}_0 端子は、64180 からモデムへ送信するときの制御信号として使用され、 \overline{DCD}_0 端子は 64180 がモデムから受信するときの制御信号として使用されます。モデム制御信号を用いた送信および受信動作を、図 6・7 を用いて説明します。

〔1〕送信動作 \overline{CTS}_0 端子は TDRE (トランスミットデータレジスタエンプティ) フラグを讀出し可能とし、かつ送信データをモデムへ送るときにの制御信号として使用されます。 $\overline{CTS}_0 = "1"$ の間は、TDRE は強制的に "0" レベルとなりますが、 $\overline{CTS}_0 = "0"$ になると TDRE は TDR₀ (ASCII トランスミットデータレジスタ) の状態に従って変化します。つまり、 $\overline{CTS}_0 = "0"$, TDRE = "1"

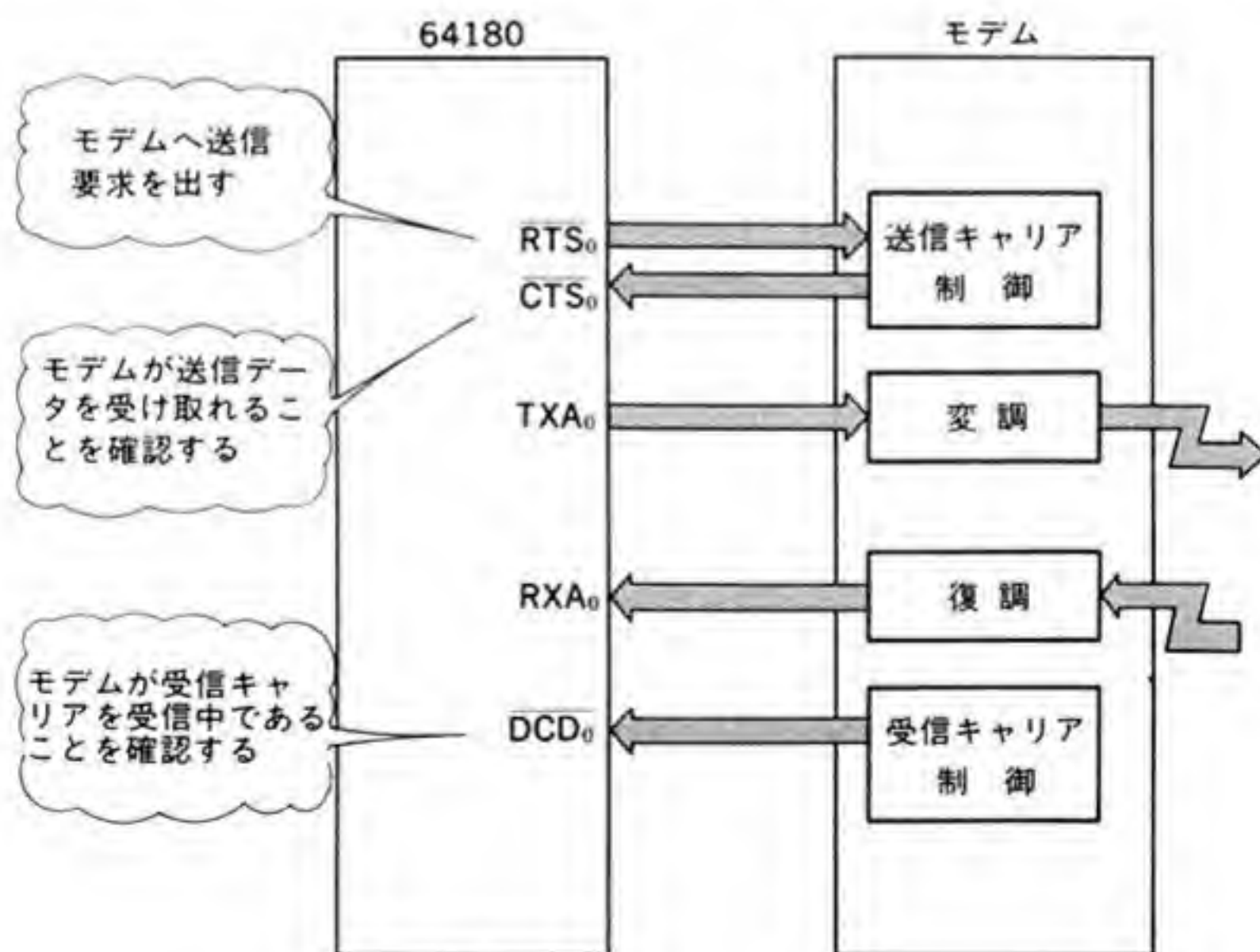


図 6・6 モデムとの接続

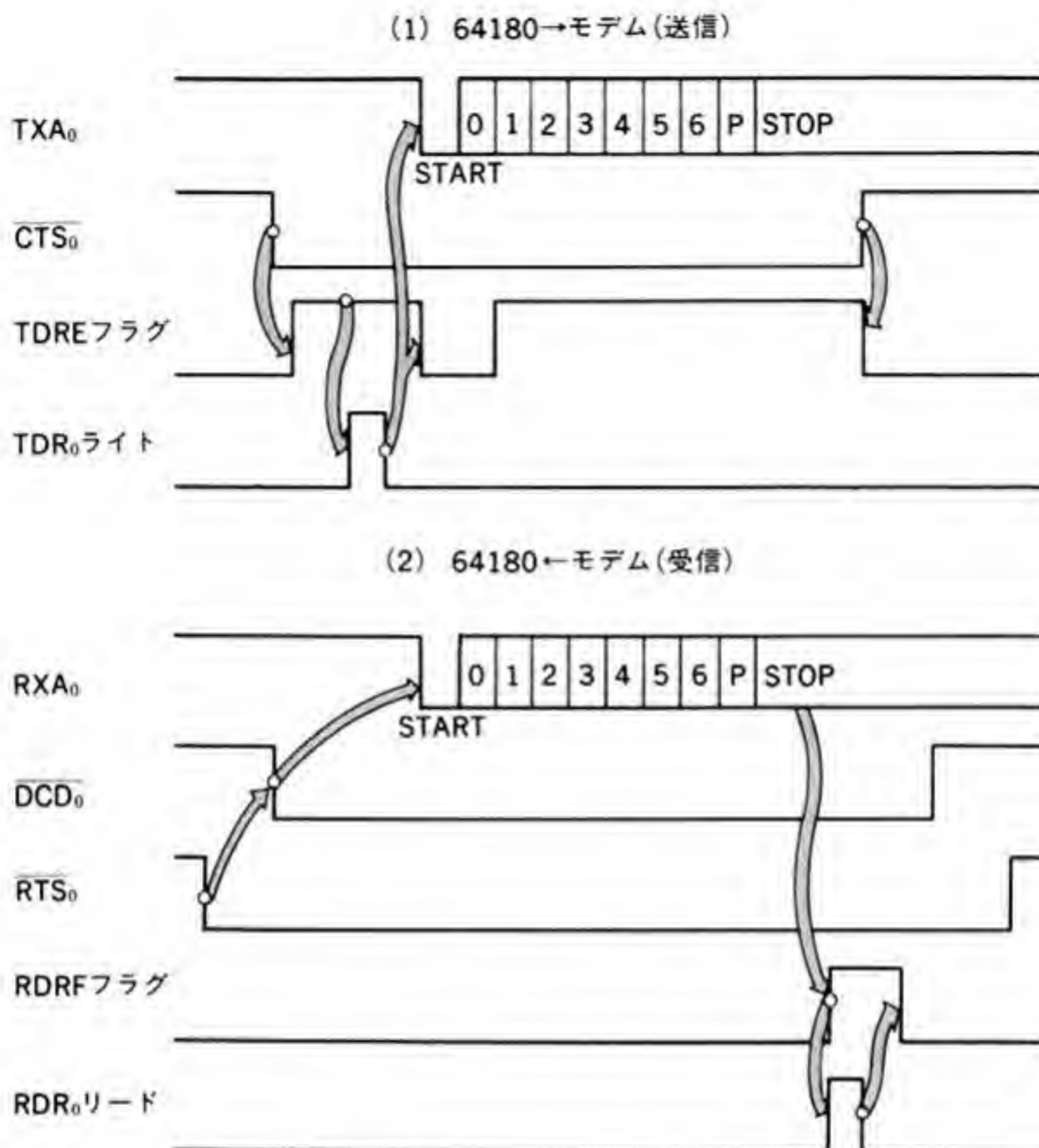


図 6・7 送/受信動作タイミング

のときに TDR_0 へ送信したいデータをライトすると、 TXA_0 端子よりデータが送信されます。送信が終了すると $\overline{CTS_0}$ 端子を“1”レベルにして処理を終わります。

〔2〕受信動作 64180 はモデムに対する送信要求のため、 RTS_0 端子を“0”レベルにクリアします。モデムから送信するデータがある場合は、 $\overline{DCD_0}$ 端子が“0”レベルとなり、 RXA_0 端子よりデータが送られてきます。 $\overline{DCD_0}$ 端子はモデムのキャリア検出を示す信号であり、“0”レベルの間は受信動作が可能です。“1”レベルになると受信動作はただちに中止されます。 $RDRF$ (レシーブデータレジスタフル) フラグが“1”になるのを確認したら、 RDR_0 (ASCII レシーブデータレジスタ) をリードし、 $\overline{DCD_0}$ = “1” となり受信処理が終了すると RTS_0 を“1”へセットします。

6・6 ASCII の使用例

ASCII の具体的応用例の中には、64180 \leftrightarrow モデムのインタフェース、64180 \leftrightarrow ACIA (HD6850/HD6350) のインタフェース、64180 \leftrightarrow UART (8251) のインタフェースが考えられます。ここでは、64180 \leftrightarrow ACIA (HD6350) のインタフェースをハードウェア、ソフトウェアの両面から説明します。

〔1〕 64180 \leftrightarrow ACIA (HD6350) とのインタフェース 64180 で ASCII と呼んでいる非同期シリアルコミュニケーションと同等機能の周辺 LSI が ACIA (Asynchronous Communication Interface Adapter) HD6350 です。64180 と ACIA は図 6・8 に示すように、RXA₀ (64180) \leftarrow TxDatA (HD6350)、TXA₀ \rightarrow RxDatA、RTS₀ \rightarrow CTS、CTS₀ \leftarrow RTS と送受信に関する信号を互いにクロスして接続します。DCD 端子は 64180 も HD6350 も内蔵していますが、モデムとの接続ではないのでともに“0”レベルに固定します。HD6350 は RxCLK と TxCLK が別々になっていますが、64180 は 1 本のクロック CKA₀ しかありませんので、送信クロック TxCLK と受信クロック RxCLK は同一の転送速度とし、CKA₀ \rightarrow RxCLK/TxCLK と接続します。

〔2〕 ソフトウェア例 図 6・9 に ACIA とのインタフェースで必要な初期設

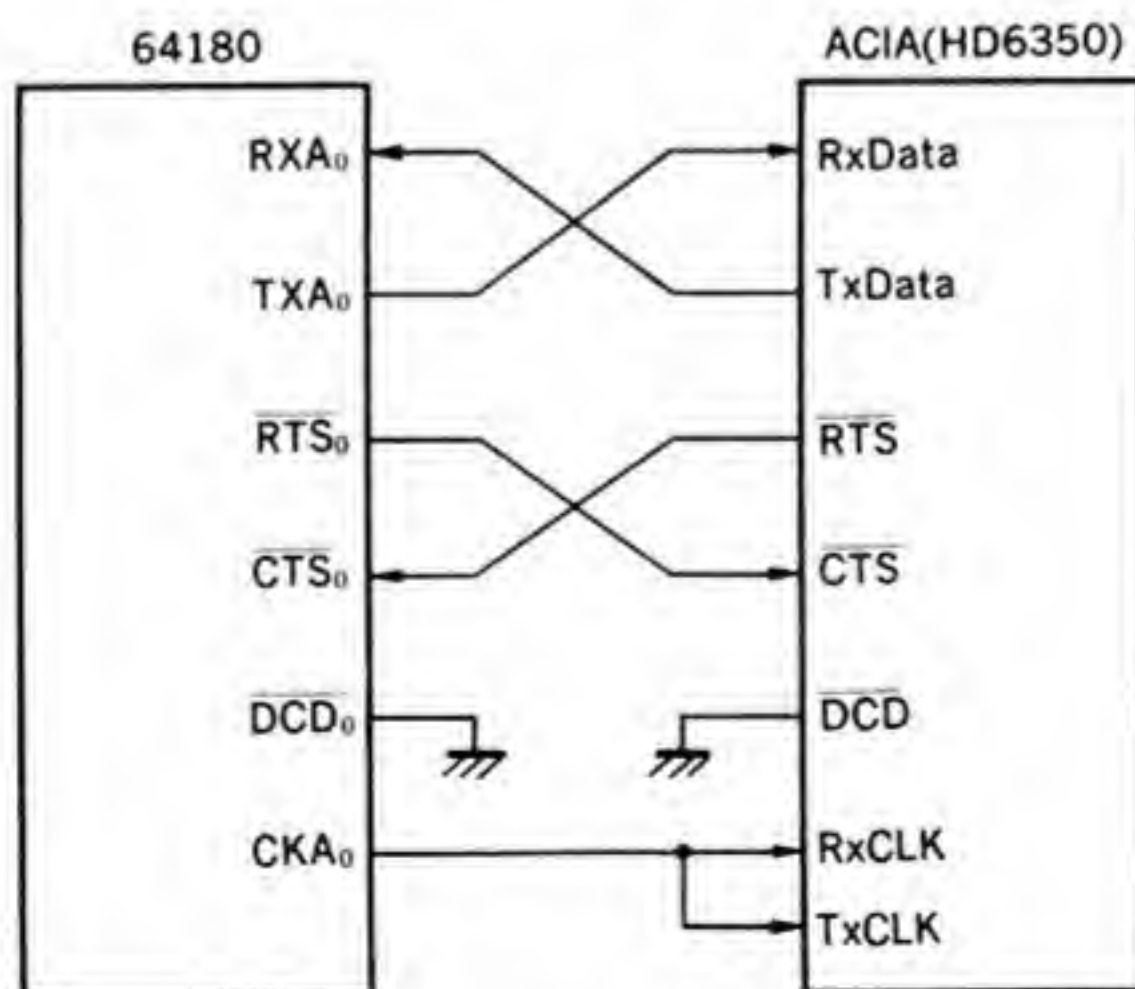


図 6・8 64180 と ACIA とのインタフェース

初期設定

LD	A, 02H	}	ボーレート 1200BPS に設定
OUTO	(CNTLB0), A		(at $\phi = 6.144\text{MHz}$)
LD	A, 64H	}	送受信許可 (TE = 1, RE = 1)
OUTO	(CNTLA0), A		8ビットデータ + 1ストップビット
LD	A, 08H	}	受信割込み許可 (RIE = 1)
OUTO	(STAT0), A		送信割込み禁止 (TIE = 0)
EI			割込み許可 (IEF = 1)

送信動作

T1: INO	A, (STAT0)	}	TDRE = "1" かチェック
BIT	1, A		
JR	Z, T1		
LD	A, XXH	}	送信データ XXH を TDRO へセット
OUTO	(TDR0), A		

受信動作

S1: INO	A, (STAT0)	}	RDRF = "1" かチェック
BIT	7, A		
JR	Z, ERROR		
AND	70H	}	フレミングエラー, オーバランエラー, パリティエラーのチェック 受信データリード
JR	NZ, ERROR		
INO	A, (RDR0)		

ERROR: エラー処理

RET

図 6・9 ソフトウェア例

定, 送信動作, 受信動作のソフトウェア例を示します。ASCIIの初期設定では, ボーレートを CNTLB0 (ASCIIコントロールレジスタ B) で設定し, 送受信許可フラグ TE/RE およびデータフォーマットを CNTLA0 (ASCIIコントロールレジスタ A₀) で設定し, RIE = "1", TIE = "0" を STAT0 (ASCIIステータスレジスタ 0) で設定後, EI 命令で割込み許可の状態にします。一方, 送信動作では TDRE = "1" か否かをチェック後, 送信したいデータを TDR0 へライトします。また, 受

6. 非同期方式シリアル I/O

信動作では RDRF = "1" か否かをチェックした後エラーチェックをし、受信データを RDR0 (レシーブデータレジスタ 0) からリードします。ソフトウェアの基本的アルゴリズムは同じですので、ここに示すソフトウェア例は他の周辺 LSI と接続するときでも使用できます。

ASCI のストップビット長

ASCI のストップビット長を 1 ビットにするか 2 ビットにするかにより、送受信がうまくいくときとうまくいかないときがあります。たとえば、64180 の ASCI が受信側で、送信側にストップビット長が 1 ビットと 2 ビットの 2 種類 ACIA があったとします。受信側のストップビットを 2 ビットで選択した場合、送信側のストップビットが 2 ビットだと正常に受信しますが、送信側のストップビットが 1 ビットだとフレミングエラーとなります。一方、受信側のストップビットを 1 ビットで選択した場合、送信側のストップビットが 1 ビットでも 2 ビットでも正常に受信します。したがって、受信側では 1 ビット、送信側では 2 ビットのストップビットを選択するのが、システム設計のノウハウです!?

7. クロック同期方式シリアル I/O

高速シリアル通信を実現するのがクロック同期方式シリアル I/O であり、マイコン間通信やマイコンと周辺 LSI 間通信に適しています。クロックや立下リエッジに同期してデータの送受信を行うため、短い距離のシリアル通信として用いられます。内部のトランスミット/レシーブデータレジスタが 1 個で送信と受信で共用しているため、半 2 重通信方式をサポートしています。

7・1 CSI/O の構成と機能

クロック同期方式シリアル I/O ポート (CSI/O) は、非同期方式シリアル I/O (ASCII) に比べ、転送速度が最大 400 kbps ($\phi = 8 \text{ MHz}$) と非常に高速です。このため、伝送距離の短い機器内通信で採用される場合が多く、マイコン \leftrightarrow マイコン、マイコン \leftrightarrow 周辺 LSI 間のデータ転送では重要な役割を果たしています。

64180 ファミリーに内蔵している CSI/O は図 7・1 に示すように、RXS、TXS、CKS 端子と、CSI/O トランスミット/レシーブデータレジスタ、および CSI/O コントロールレジスタから構成されています。

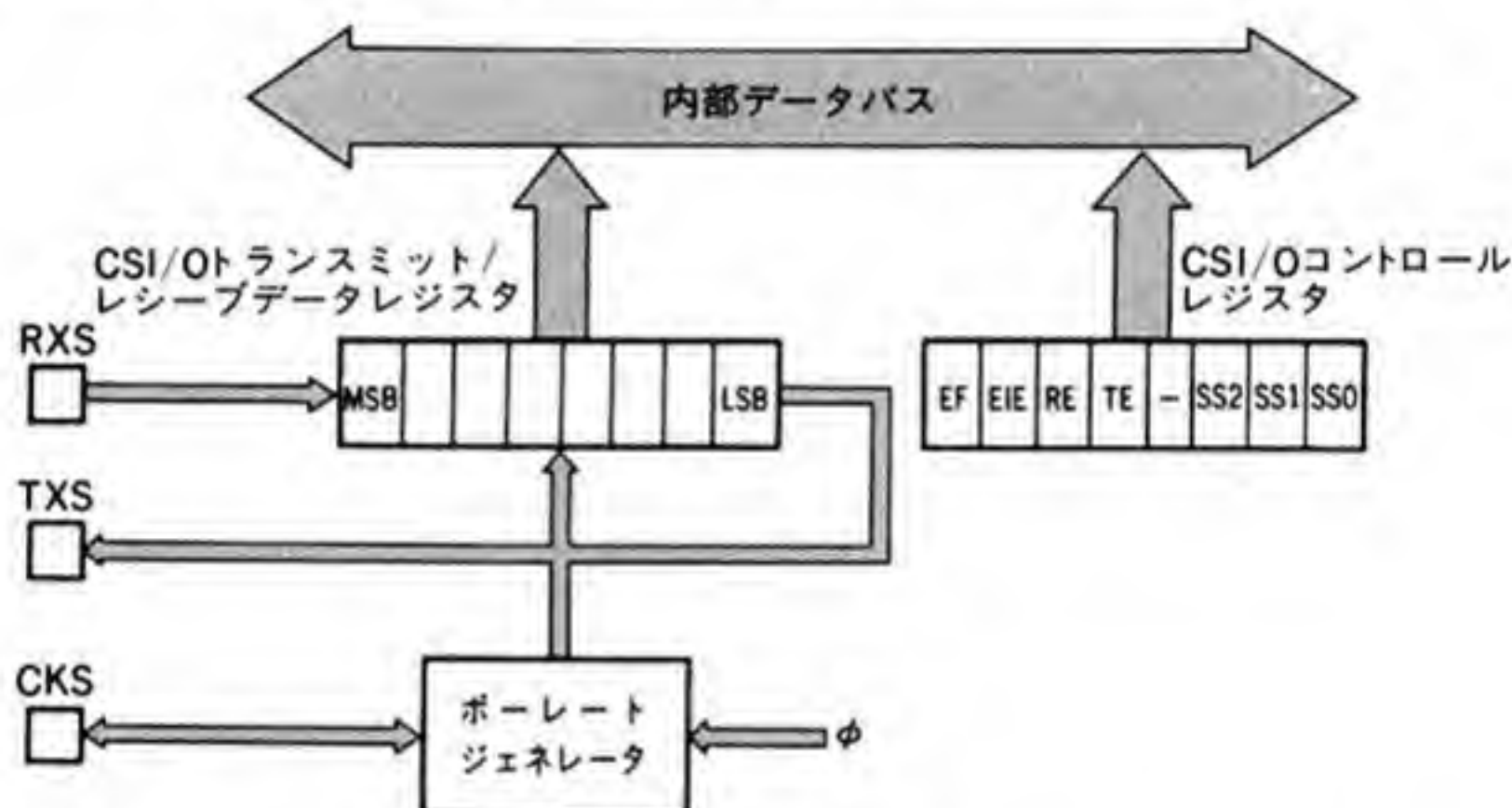


図 7・1 CSI/O ブロック図

〔1〕 **CSI/O トランスミット/レシーブデータレジスタ** RXS 端子（受信端子）あるいは TXS 端子（送信端子）を経由して入出力するデータが、CSI/O トランスミット/レシーブデータレジスタへ入ります。レジスタは 1 本ですので、同時に送信、受信は行えません。つまり半 2 重通信方式をサポートしています。また、バッファレジスタをもっていないため、送信途中でこのレジスタにライトすると送信データは変化します。同様に、受信途中でこのレジスタにデータをライトしても受信データは破壊しますので、アプリケーションソフトウェアで制御しなければなりません。

〔2〕 **CSI/O コントロールレジスタ** CSI/O コントロールレジスタの内容は、ボーレートの選択ビット (SS0～SS2)、トランスミットイネーブルフラグ

表 7・1 ボーレートの設定

CSI/O コントロールレジスタ			ボーレート
SS2	SS1	SS0	
0	0	0	$\phi \div 20$
0	0	1	$\phi \div 40$
0	1	0	$\phi \div 80$
0	1	1	$\phi \div 160$
1	0	0	$\phi \div 320$
1	0	1	$\phi \div 640$
1	1	0	$\phi \div 1\,280$
1	1	1	外部クロック入力

(TE)、レシーブイネーブルフラグ (RE)、エンドフラグ (EF) およびエンドインタラプトイネーブル (EIE) です。

ボーレートは表 7・1 に示すように、SS0～SS2 (Speed Select) ビットで外部または内部 7 種類 ($\phi \div 20$, $\div 40$, $\div 80$, $\div 160$, $\div 320$, $\div 640$, $\div 1\,280$) から選択できますので、 $\phi = 8\text{ MHz}$ だと最高 400 kbps となります。

送信の開始はトランスミットイネーブルフラグで制御し、受信開始はレシーブイネーブルフラグで制御できます。また、送受信が完了したことを示すエンドフラグがあり、エンドフラグが“1”のとき CPU に割込み要求を許可するエンドインタラプトイネーブルフラグがありますので、シリアル通信を用途に合わせてソフトウェアで制御できます。

7・2 CSI/O の動作原理

CSI/O の送受信の動作原理を図7・2 および図7・3を用いて説明します。

〔1〕送信動作 送信動作を始めるときは、TE(トランスミットイネーブル)フラグが“0”であるかを確認します。TE=“1”は前回の送信データが完了していないことを意味しますので、TE=“0”になるまで待たなければなりません。TE=“0”になったら、送信したいデータをCSI/Oトランスミットレシーブデータレジスタへライトします。この後でTE=“1”にセットします。TE=“1”になると送信データがTXS 端子からCKS クロックの立下りエッジに同期して出力されます。すなわち、内部クロックでボーレートを作成している場合

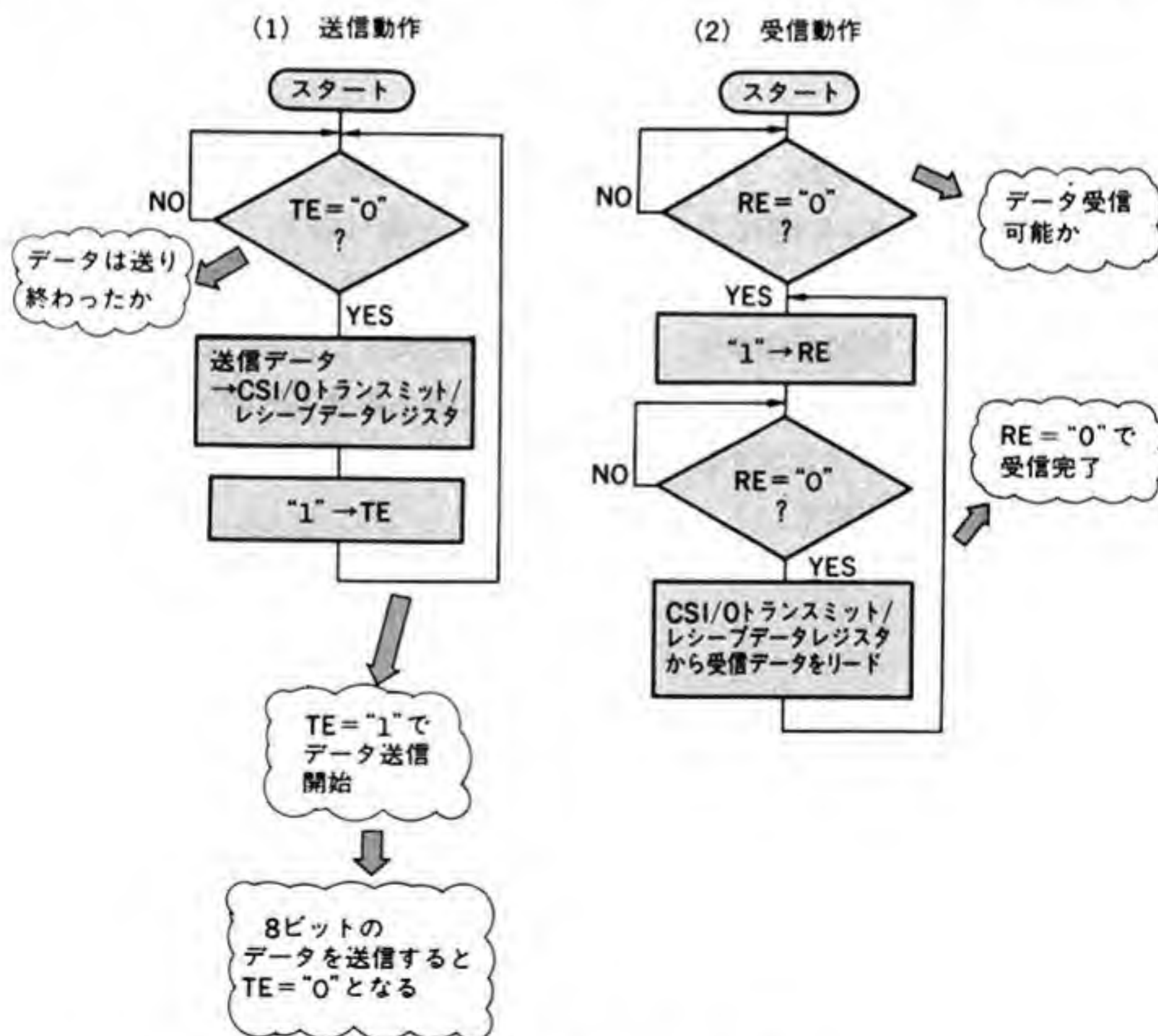


図 7・2 送/受信フローチャート

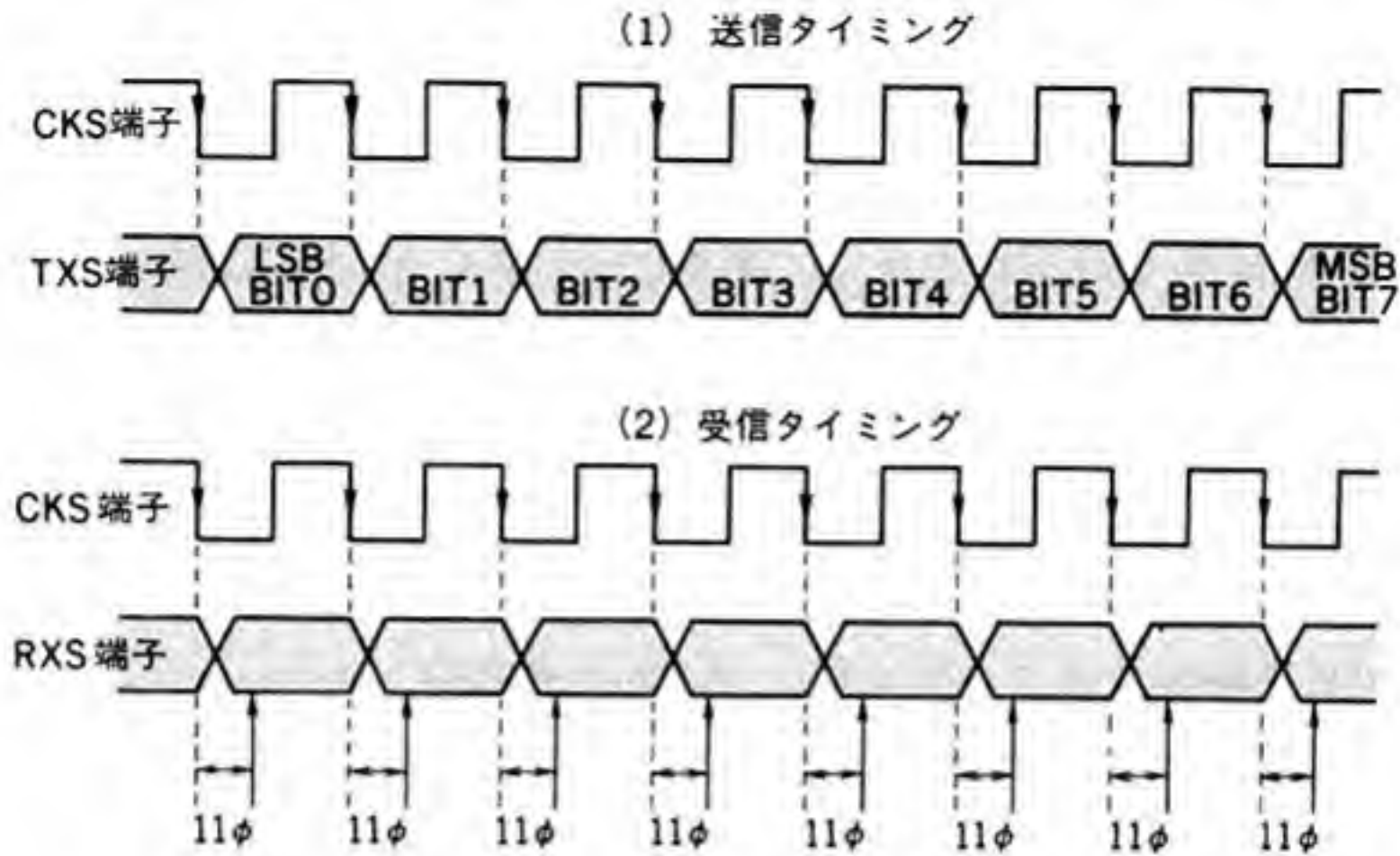


図 7・3 送/受信タイミング

は、CKS 端子からクロックが8周期分出力され、このクロックに同期して送信データが TXS 端子より出力されます。一方、外部クロックでボーレートを作成している場合は、CKS 端子より外部クロックが入力されると、それに同期してデータ送信されます。8ビットのデータ送信終了後は $TE = "0"$ にクリアされますので、次のデータを送ることができます。これらの処理の繰返して送信を続けることができます。

〔2〕受信動作 受信動作を始めるときは、**RE**（レシーブイネーブル）ビットが“0”であるか確認します。 $RE = "1"$ は前回のデータの受信が完了していないことを意味しますので、 $RE = "0"$ になるまで待ちます。 $RE = "0"$ の後、 $RE = "1"$ にすると受信を開始します。つまり、送信動作同様 CKS 端子のクロックに同期して RXS 端子の状態が CSI/O データレジスタに取り込まれ、8ビットの受信が終了すると $RE = "0"$ にクリアされます。 $RE = "0"$ を確認したら CSI/O トランスミット/レシーブデータレジスタから受信データをリードします。次のデータを受信するときは、 $RE = "1"$ にすることにより繰返し受信処理を続けることができます。

7・3 CSI/O の使用例

CSI/O の具体的な使用例として、①4ビットマイコン HMCS400 シリーズとのインタフェース、②LCD ドライバ HD61100 とのインタフェースを示します。

〔1〕 4ビットマイコン HMCS400 とのインタフェース 4ビットマイコン HMCS400 シリーズは、64180 と同等機能のクロック同期方式シリアル I/O を内蔵していますので、図 7・4 に示すように TXS (64180) → SI (HMCS400), RXS ← SO, CKS ⇄ SCK を接続することにより、マイコン間のインタフェースが可能となります。HMCS400 シリーズは I/O ポート、LCD ドライバ、蛍光表示管ドライバなど豊富な機能を内蔵していますので、64180 に内蔵していない機能を拡張したい場合に有効です。

〔2〕 LCD ドライバ HD61100 とのインタフェース マイコン周辺 LSI でもクロック同期方式シリアル I/O を内蔵しているデバイスがたくさんあります。たとえば、LCD ドライバ HD61100 と 64180 を接続する場合、図 7・5 に示すように

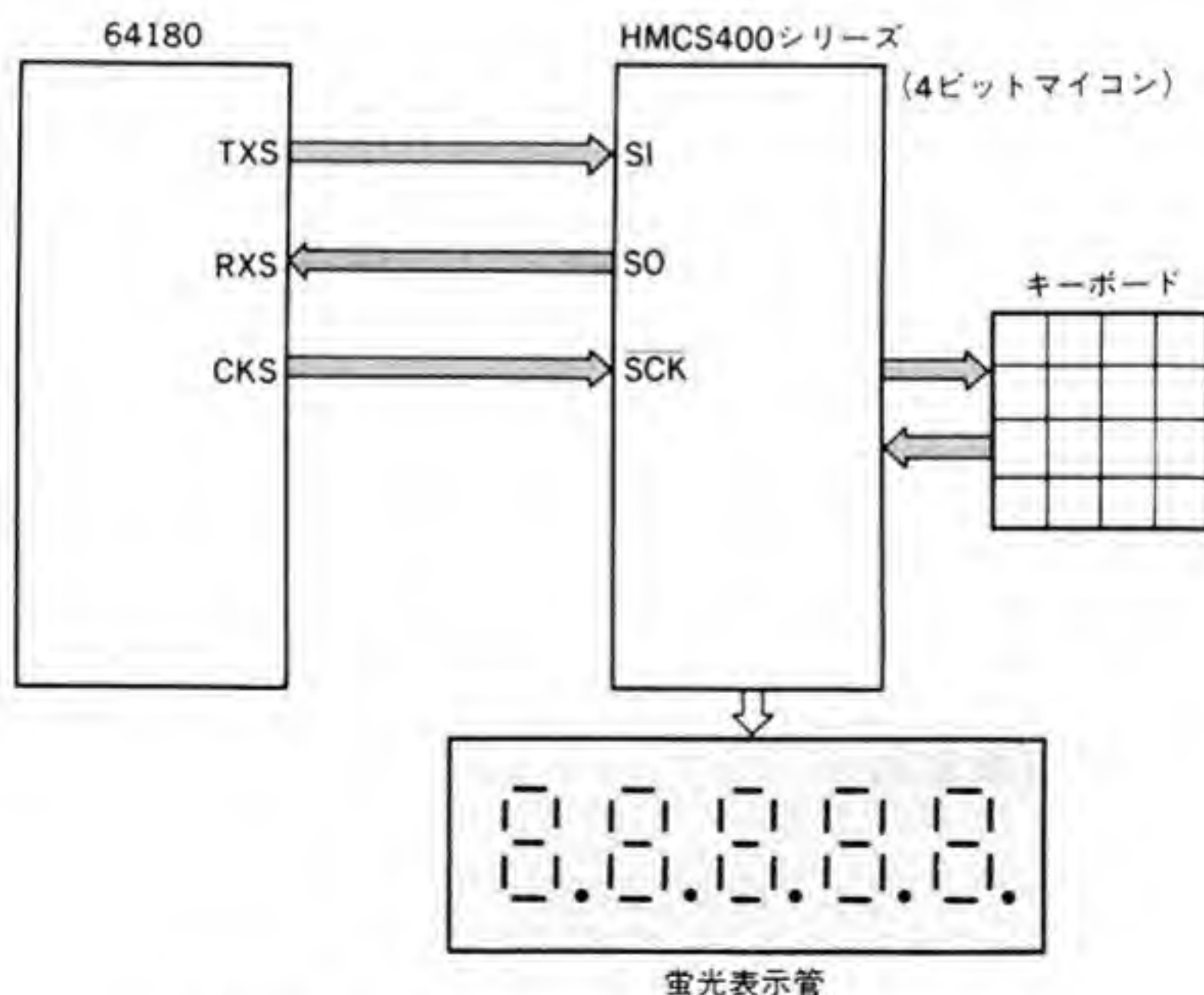


図 7・4 64180 と 4 ビットマイコンとのインタフェース

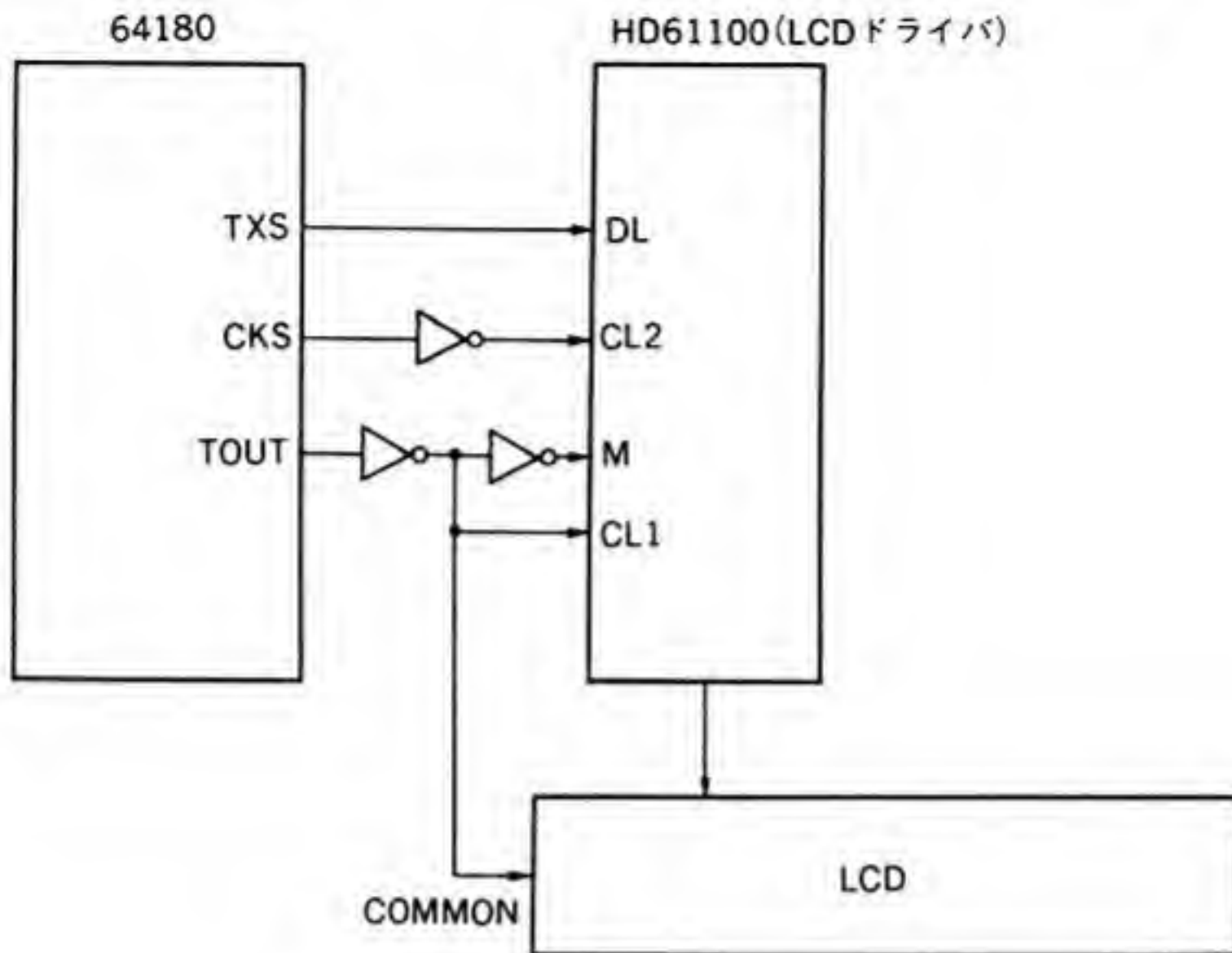
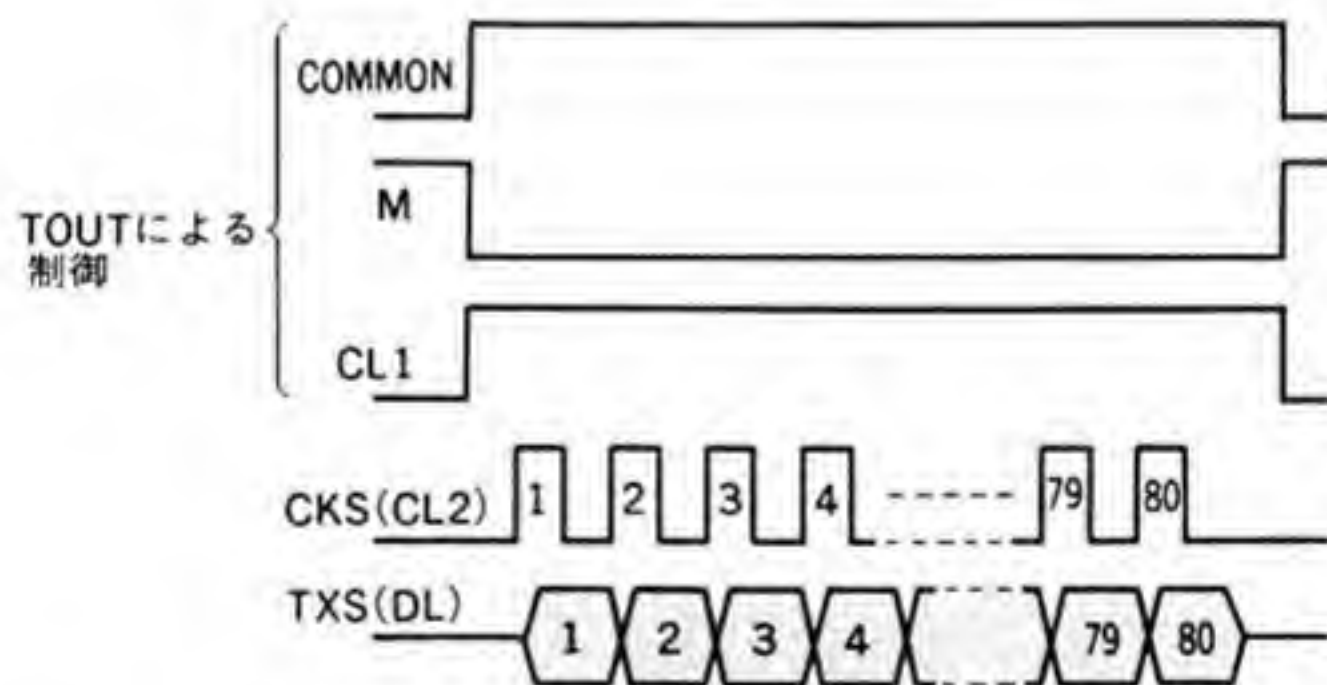


図 7・5 64180 と LCD ドライバとのインタフェース



LCDDP: LD A,(HL) 送信したいセグメントデータのロード
 INO C,(CNTR) CNTRのリード
 BIT TE,C TE="0"のチェック
 JR NZ,LCDDP TE="1"だとLCDDPへ
 OUTO (TRDR),A TRDRへデータをライト
 LD A,10H
 OUTO (CNTR),A } TE="1"をセットし送信開始

図 7・6 64180 と HD61100 のインタフェースタイミング
およびソフトウェア

7. クロック同期方式シリアルI/O

TXS(64180)→DL(HD61100), CKS→CL2を接続するとクロック同期方式のシリアル通信ができます。また、HD61100はLCDを表示するための交流信号MとラッチクロックCL1が必要ですので、これには64180のタイマ出力TOUTを用います。

64180から出力する信号のタイミングは、図7・6に示すようにTOUT, CKS, TXSで決まります。このうち、CSI/Oで制御するCKS端子からのクロック送出およびTXS端子からのデータ送信の手順を、ソフトウェア例をみながら説明します。

手順は、①送信したいセグメントデータのロード、②TEビットのチェック、③TRDR(トランスミット・レシーブ・データ・レジスタ)へ送信データのライト、④TE="1"にセットとなります。このように7ステップという短いプログラムで8ビットのデータ送信が可能となります。80セグメントのLCDを表示する場合は、この基本ルーチンを10回繰り返しCL1を立ち下げることにより表示データが変更可能となります。

8. WAIT リフレッシュ コントローラ

64180 の CPU は低速のメモリや I/O 機器をアクセスする場合に、バスサイクルを引き延ばしています。また、DRAM とインタフェースするためには、リフレッシュが必要です。従来、これらの機能は外付回路にて実現していましたが、64180 にはこれらのいわゆる雑ロジックを内蔵しているため、メモリや I/O とのインタフェース回路が簡単になります。

8・1 ウェイトの動作原理と使用法

〔1〕 概 要 64180 には低速のメモリや I/O のアクセスのために、リードまたはライトサイクルを引き延ばすためのウェイトステート挿入機能があります。ウェイトは 1 クロック単位で与え、 T_2 ステートと T_3 ステートとの間にダミーのステートとしてウェイトステート T_w が挿入されて、そのサイクルを引き延ばしています。ウェイトステートの挿入は、ハードウェアによる方法 ($\overline{\text{WAIT}}$ 端子への 0 レベル入力) とソフトウェアによる方法 (ウェイトコントロールレジスタのビット設定) とがあります。

〔2〕 ハードウェアによる方法 この方法は、図 8・1 に示すように T_2 ステートの ϕ クロックの立下りのタイミングで、 $\overline{\text{WAIT}}$ 端子の状態をサンプリングし、次のように動作します。

$\overline{\text{WAIT}}$ 端子が “L” のとき…… T_w ステートが挿入される

$\overline{\text{WAIT}}$ 端子が “H” のとき…… T_w ステートは挿入されず

T_3 ステートへ移る

引き続いて T_w ステートを 2 個以上挿入する場合は、 $\overline{\text{WAIT}}$ 端子を必要な数だけ “L” にします。



図 8・1 $\overline{\text{WAIT}}$ 端子によるウェイトステートの挿入

〔3〕 ソフトウェアによる方法 ウェイトステートコントロールレジスタのビットを設定することにより、規定数の T_w ステートを挿入する方法です(図 8・2 参照)。図のように、メモリ空間と I/O 空間とは独立して T_w ステート数を設定できるので、それぞれのアクセスタイムに最適な T_w ステート数が選べます。メモリ空間では 0～3、I/O 空間では 1～4 です。 T_w ステート数は、それぞれの空間の

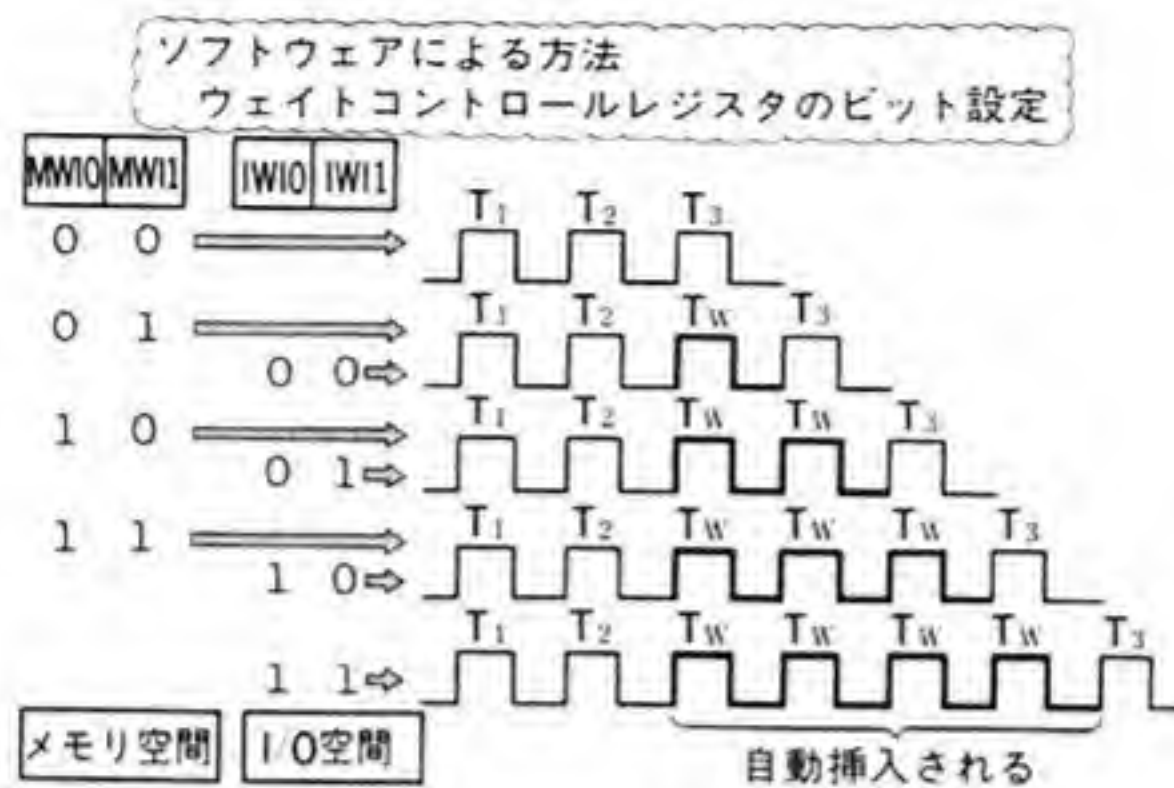


図 8・2 ウェイトコントロールレジスタによる
ウェイトステートの挿入

中では一定です。したがって、たとえばアクセスタイムの遅い EPROM と速い SRAM が混在する場合、EPROM に合わせて T_w ステート数を設定することになります。ウェイトステートコントロールレジスタは、リセット後には最大 T_w ステート数となるように初期化されます。また、そのレジスタは DMAC チャンネル 1 のコントロールレジスタと I/O アドレスが同じです。

〔4〕 **ソフトウェアとハードウェアの同時要求** T_w ステート挿入要求がソフトウェアとハードウェアともに同時の場合には、長いほうに合わせて挿入されます。

8・2 リフレッシュコントローラの動作原理

〔1〕 概 要 64180では、DRAM リフレッシュはリフレッシュサイクルという特別なマシンサイクルにて行われます。Z80ではオペコードフェッチサイクルに付随していますが、64180ではオペコードフェッチサイクルとは独立しています。また、64180ではリフレッシュサイクルを挿入するかしないか、リフレッシュサイクルの挿入の間隔、リフレッシュサイクル長を2ステートか3ステートかを選択できます。

〔2〕 リフレッシュサイクル マシンサイクルの切れ目で挿入されます。図8・3の例では命令実行中に挿入されていますが、DMA 実行中でもマシンサイクルの切れ目に挿入されます。図8・3のリフレッシュ要求の間隔は10ステートを選択しています。しかしながら、図8・3では「OUTO (3F), A」命令のI/Oライトサイクル後に挿入されるため、10ステートより長くなったり、あるいは短くなったりします。一方、リフレッシュサイクルが挿入されないのは次の場合です、リ

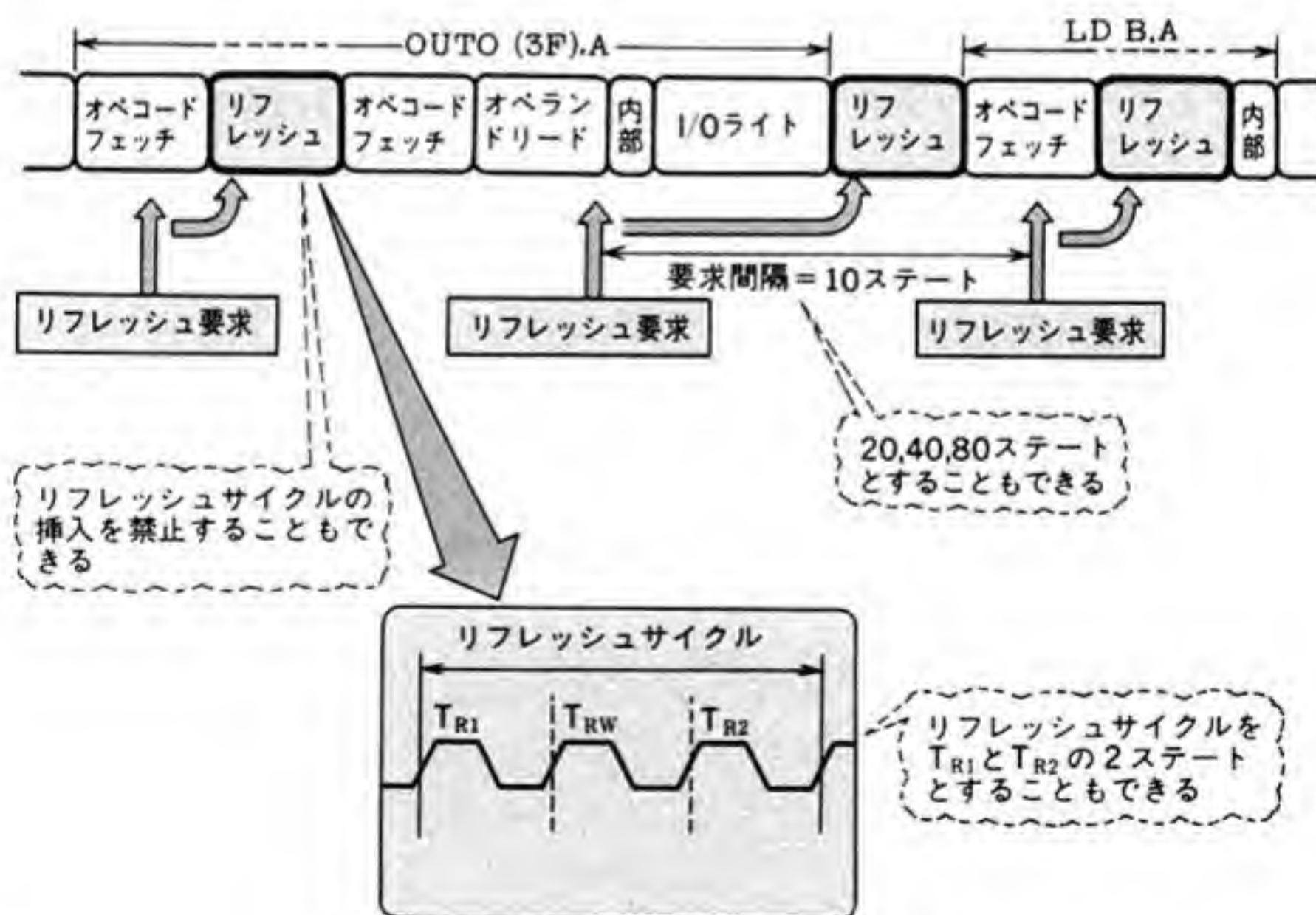


図 8・3 リフレッシュサイクルの挿入動作

8・2 リフレッシュコントローラの動作原理

セット、バスリリースモード、スリープおよびシステムストップモード、ウェイトステートが長く挿入されている場合です。このようなモードのときに DRAM のデータが消失しては困る場合には、別の手段でリフレッシュされるような工夫をします。

〔3〕 リフレッシュアドレス $A_0 \sim A_7$ 64180 では 8 ビットです。256K DRAM のような 256 リフレッシュサイクルの DRAM には、外部回路を付加せずに使用できます。なお、CPU の専用レジスタの R カウンタは Z80 ではリフレッシュアドレスですが、64180 ではリフレッシュアドレスとは一致していません。オペコードフェッチごとにカウントアップするカウンタです。

〔4〕 リセット後 リフレッシュアドレスは 00H からスタートして、自動的にリフレッシュサイクルが挿入されます。挿入間隔は最短の 10 ステートで、リフレッシュサイクルは 3 ステートです。その後、動作周波数とメモリリフレッシュ間隔により最適値に設定することになります。また、DRAM を使っていなければリフレッシュサイクルの挿入を禁止します。

8・3 リフレッシュタイミングと DRAM とのインタフェース

〔1〕 **タイミグ** リフレッシュサイクルは、 T_{R1} 、(T_{RW})、 T_{R2} の 2 または 3 ステートで構成されています。 T_{RW} はリフレッシュウェイトステートで、レジスタ設定によりソフトウェアで挿入することができます (図 8・4)。

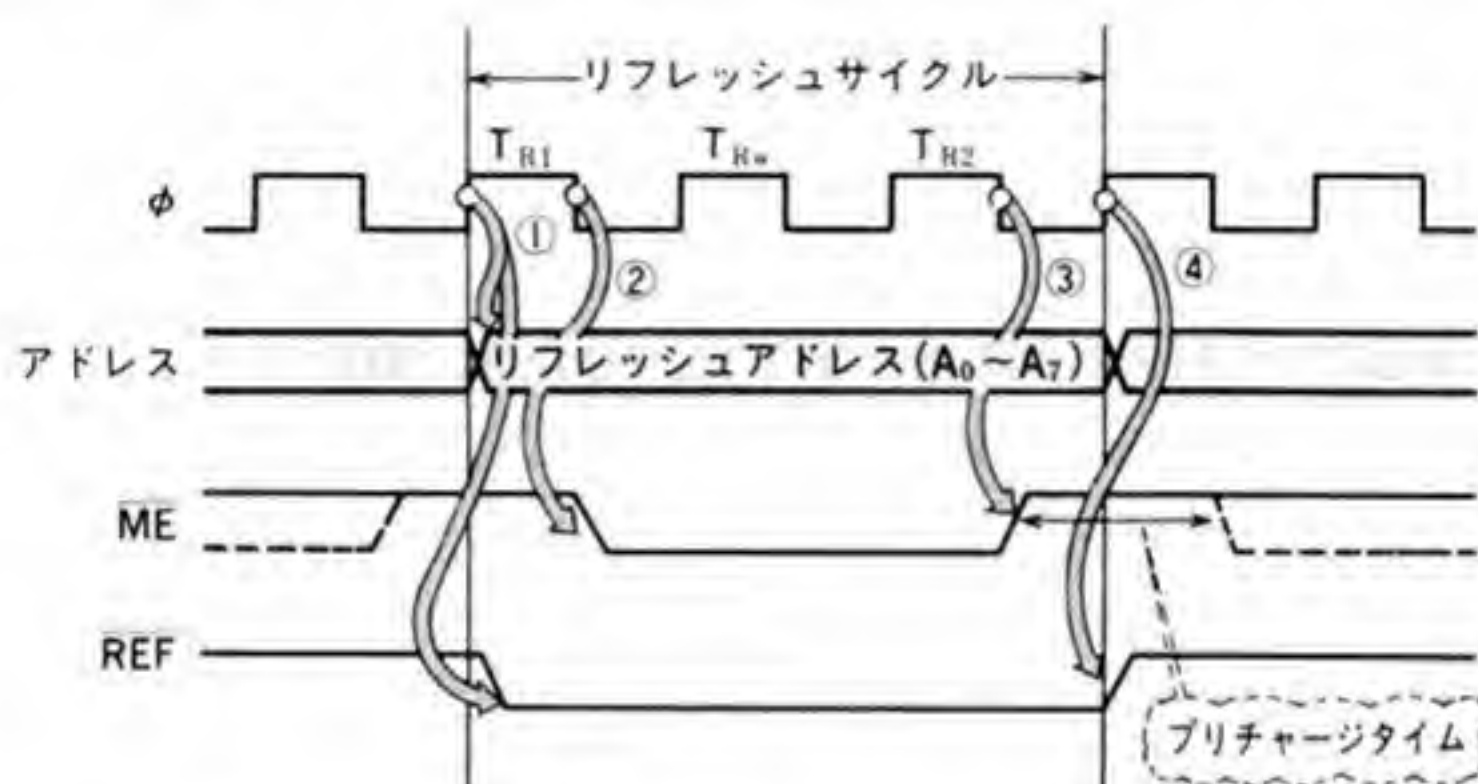


図 8・4 リフレッシュサイクル (リフレッシュウェイトを挿入している場合)

このサイクルのシーケンスは次のとおりです。①リフレッシュ T_1 ステート (T_{R1}) の前半で、リフレッシュアドレスが出力されます。同時に、 \overline{REF} 信号は“L”となります。② T_{R1} ステートの後半で、 \overline{ME} はアクティブとなりリフレッシュの起動がかかります。③リフレッシュ T_2 ステート (T_{R2}) の後半で、 \overline{ME} はインアクティブとなりますが、リフレッシュアドレスは次のバスサイクルまで保持されています。④ \overline{REF} 信号は次のサイクルの先頭でインアクティブとなります。

〔2〕 **DRAM とのインタフェース** 64180 はリフレッシュコントローラを内蔵しているため、図 8・5 のように DRAM のデータ、 \overline{RAS} 、 \overline{WE} 端子には直結できます。一方、メモリアドレスと \overline{CAS} 信号はそれぞれ外部回路で生成します。具体的なインタフェース例については、第 III 編を参照してください。

8・3 リフレッシュタイミングと DRAM とのインタフェース

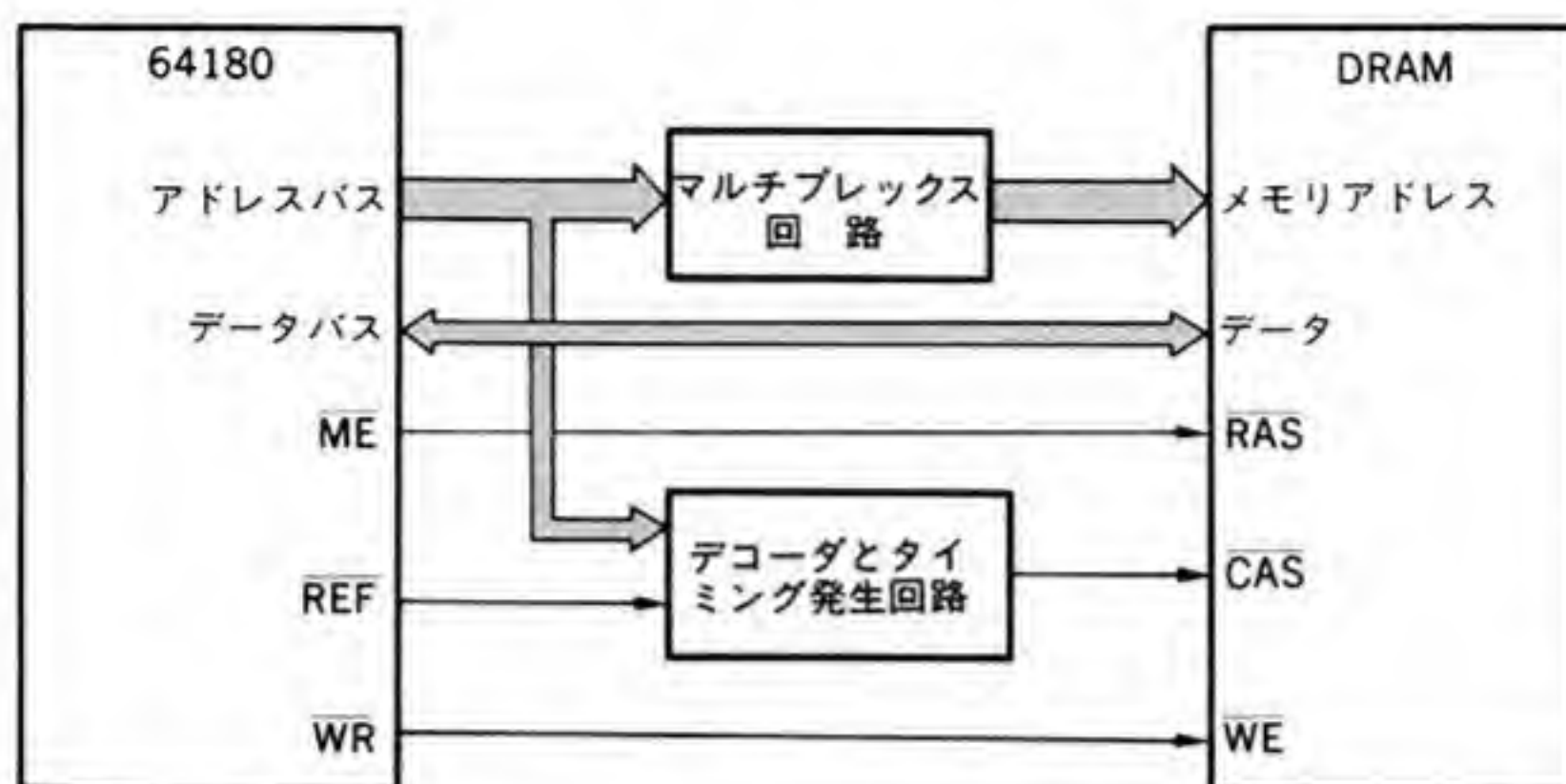


図 8・5 DRAM との接続例

9. 割 込 み

64180 は強力な割込み機能をもっています。割込み要因としては、外部 4 本、内部 8 本の計 12 本の要因があります。このうち、外部割込みは Z80 がもっていた $\overline{\text{NMI}}$ と $\overline{\text{INT0}}$ で、これに加え 64180 で追加された $\overline{\text{INT1}}$ 、 $\overline{\text{INT2}}$ の 2 本の割込みがあります。また、内部割込みは Z80 にはなかった割込みで内蔵の周辺 (DMAC, タイマ, CSI/O, ASCI) の割込みおよび TRAP 割込みです。

9・1 割 込 み と は

〔1〕 **割込みの概念** 割込みとは、通常のシーケンシャルに実行するプログラムより優先度が高く、かつ緊急に処理する必要のあるプログラムを優先して実行するための手法です。CPUは、割込みの要求があるとそこまで実行していたプログラムを中断し、割込みの要求に応じたプログラムを実行します(図9・1)。

また、割込みには、ソフトウェアでマスクできないノンマスクابل割込み(要求が入ると必ず受け付けられる割込み)と、割込みを受け付けるかどうかをソフトウェアで制御可能なマスクابل割込みがあります。

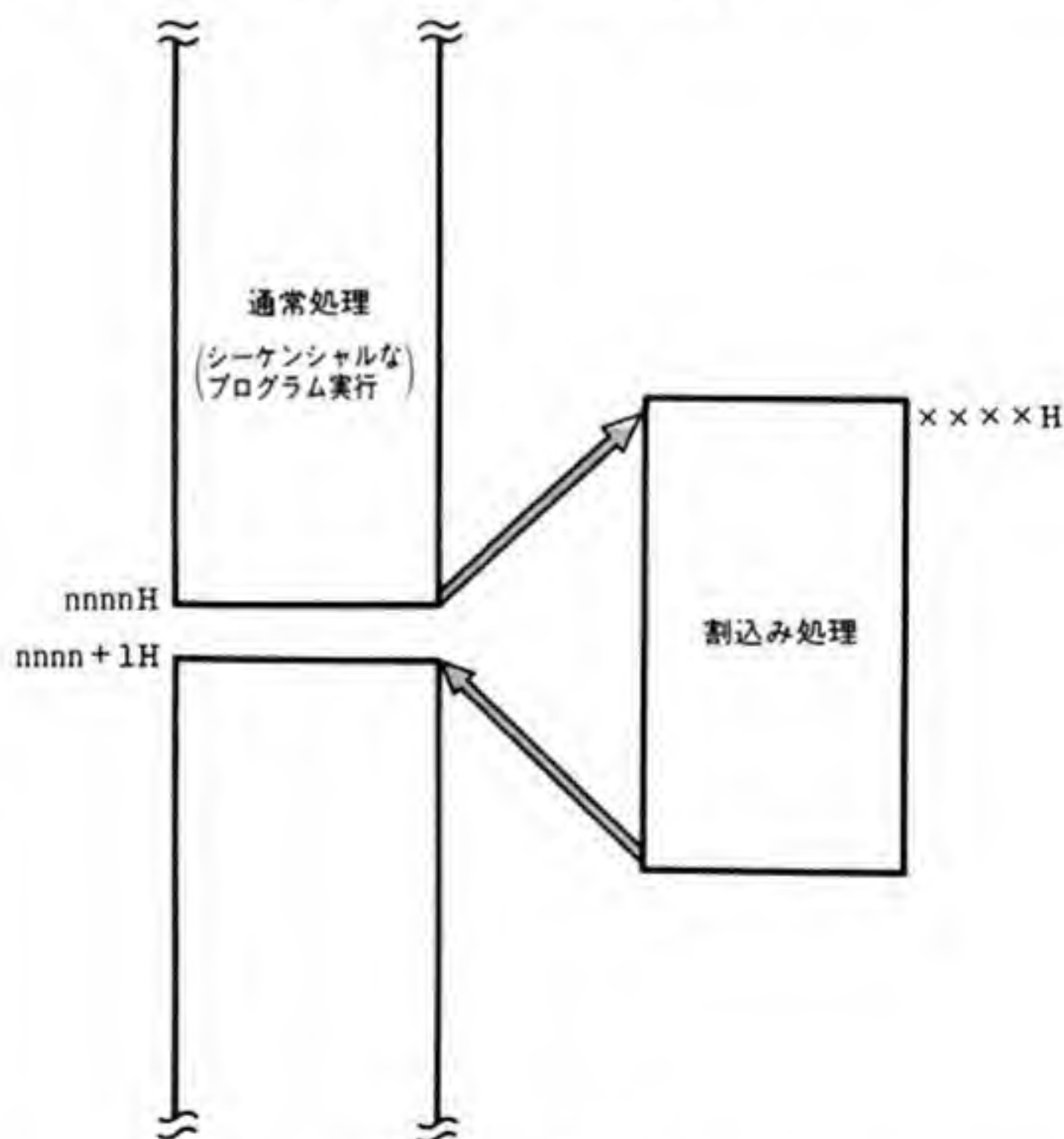


図 9・1 割込み処理

〔2〕 **64180 の割込み要因** 64180 の割込み要因としては、外部(割込みの要因が 64180 の外部に入り割込み端子から要求が入る) 4 本と、内部(割込みの要因が 64180 の内部にある) 8 本の計 12 本の要因があります。

9・1 割 込 み と は

外部割込みの要因としては、ノンマスカブルな $\overline{\text{NMI}}$ があります。 $\overline{\text{NMI}}$ が発生すると 66 番地へジャンプします。これ以外に、マスカブルな $\overline{\text{INT}}$ 3 本 ($\overline{\text{INT0}}$, $\overline{\text{INT1}}$, $\overline{\text{INT2}}$) があります。このうち $\overline{\text{INT0}}$ は、Z80 がもっていたモード 0～3 の 3 つの割込みモードをもっています。 $\overline{\text{INT1}}$ および $\overline{\text{INT2}}$ は 64180 で追加された外部割込み要因で、ベクタ方式の割込みです。

次に内部割込み要因としては、TRAP、タイマ 2 本、DMAC 2 本、CSI/O 1 本、ASCI 2 本の計 8 本があります。このうち TRAP 割込みは、未定義のオペコードをフェッチすると割込みが発生するシステムのフェールセーフのための機能です。TRAP が発生すると 0 番地へのジャンプが発生します。なお TRAP はソフトウェアでマスクできません。

また、タイマ (2)、DMAC (2)、CSI/O、ASCI (2) の計 7 つの要因は、 $\overline{\text{INT1}}$, $\overline{\text{INT2}}$ と同様なベクタ方式の割込みです。

なお、これらの割込みには優先順位があり、表 9・1 に示すとおり、TRAP, $\overline{\text{NMI}}$, $\overline{\text{INT0}}$, $\overline{\text{INT1}}$, $\overline{\text{INT2}}$, タイマ 0, タイマ 1, DMAC0, DMAC1, CSI/O, ASCI0, ASCI1 の順番の優先度となっています。

表 9・1 割込みの種類と優先順位

優先順位	割込み要因	内部/外部	備 考
1	TRAP	内 部	ベクタ (0000H 固定)
2	NMI	外 部	ベクタ (0066H 固定)
3	INT0		モード0~2
4	INT1		ベクタ方式(プログラマブル)
5	INT2		
6	タイマ0		
7	タイマ1		
8	DMAチャネル0		
9	DMAチャネル1		
10	CSI/0	内 部	
11	ASCIチャネル0		
12	ASCIチャネル1		

9.2 ベクタ方式

〔1〕 **ベクタ方式** ベクタ方式とは、割込みの処理開始番地を決定するための手法です。ベクタ方式の割込みの場合、まず割込み要因ごとに割り当てられた割込みベクタアドレスのメモリからリードを行います。この割込みベクタアドレスには、割込み処理ルーチンの開始アドレスが2バイトにわたって格納されており、CPUはこの開始アドレスから割込み処理ルーチンを実行します。したがってベクタアドレスに配置されているこの割込み処理ルーチンの開始アドレスを設定することにより、メモリ空間のすべてのエリアに割込み処理ルーチンをおくことが可能となります（図9・2）。

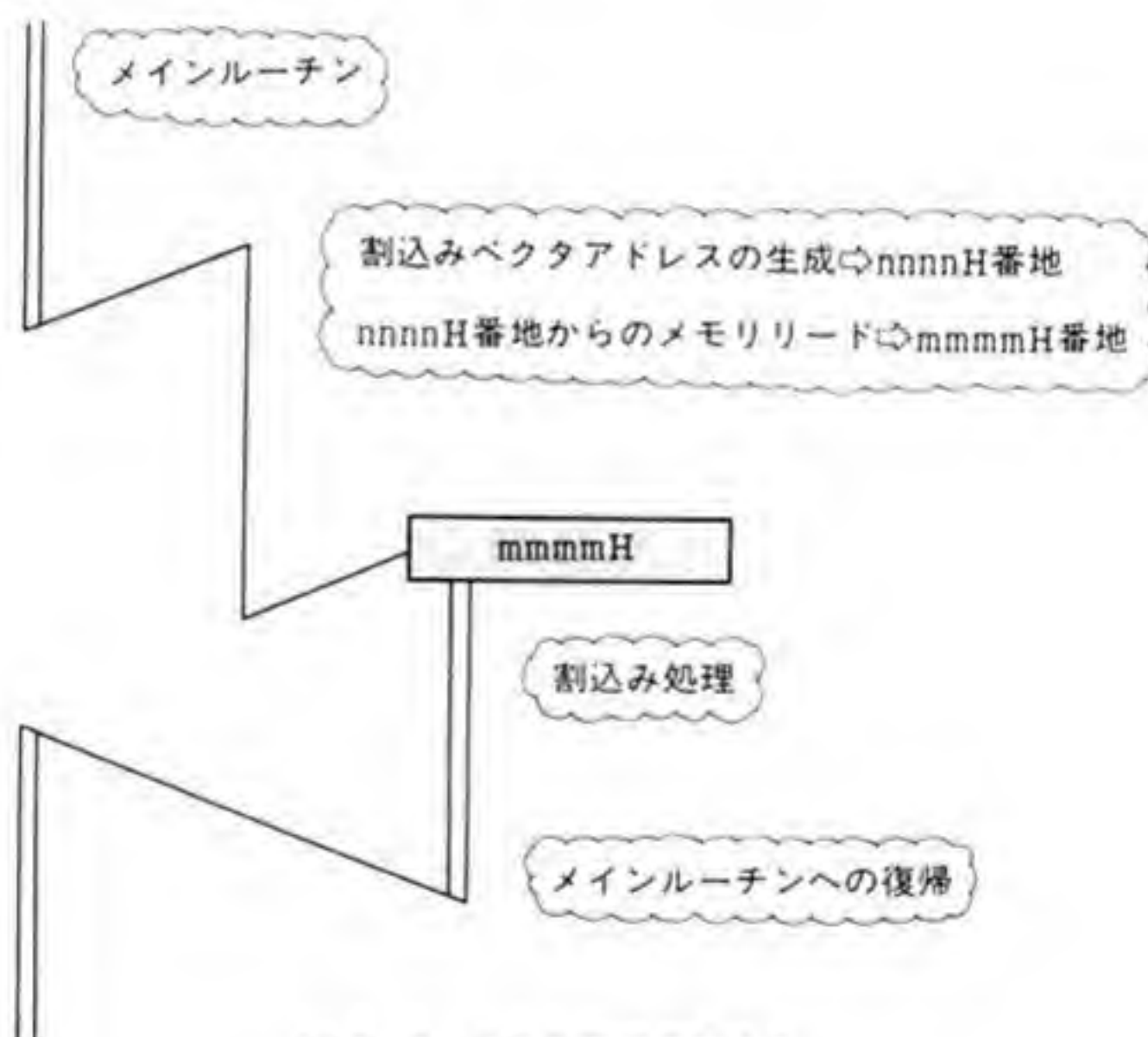


図 9・2 ベクタ方式の割込み

〔2〕 **64180 のベクタ割込み** 64180 の割込みのうち $\overline{\text{INT0}}$ （モード2）、 $\overline{\text{INT1}}$ 、 $\overline{\text{INT2}}$ 、TRAP を除く内部割込み（タイマ、DMAC、CSI/O、ASCII）は、ベクタ方式の割込みを行います。

$\overline{\text{INT0}}$ モード2のベクタは、上位8ビットにIレジスタを使用し、下位8ビットに割込みアクノレッジサイクル中に、データバスから読み込んだ8ビットのデータを使用します。

次に、 $\overline{\text{INT1}}$ 、 $\overline{\text{INT2}}$ および内部割込み（TRAP を除く）の割込みベクタアドレスは次のように生成されます。上位8ビットはIレジスタにより与えられます。下位8ビットのうち、その上位3ビットはILレジスタ（I/O空間の33H番地に配置）により与えられ、下位5ビットは要因ごとに固定コードとして与えられます。ベクタアドレスの生成の様子と固定コードを図9・3に示します。

こうして生成された16ビットのベクタアドレスから割込みルーチンの開始番地のリードを行い、おのこの割込み処理ルーチンを実行します。

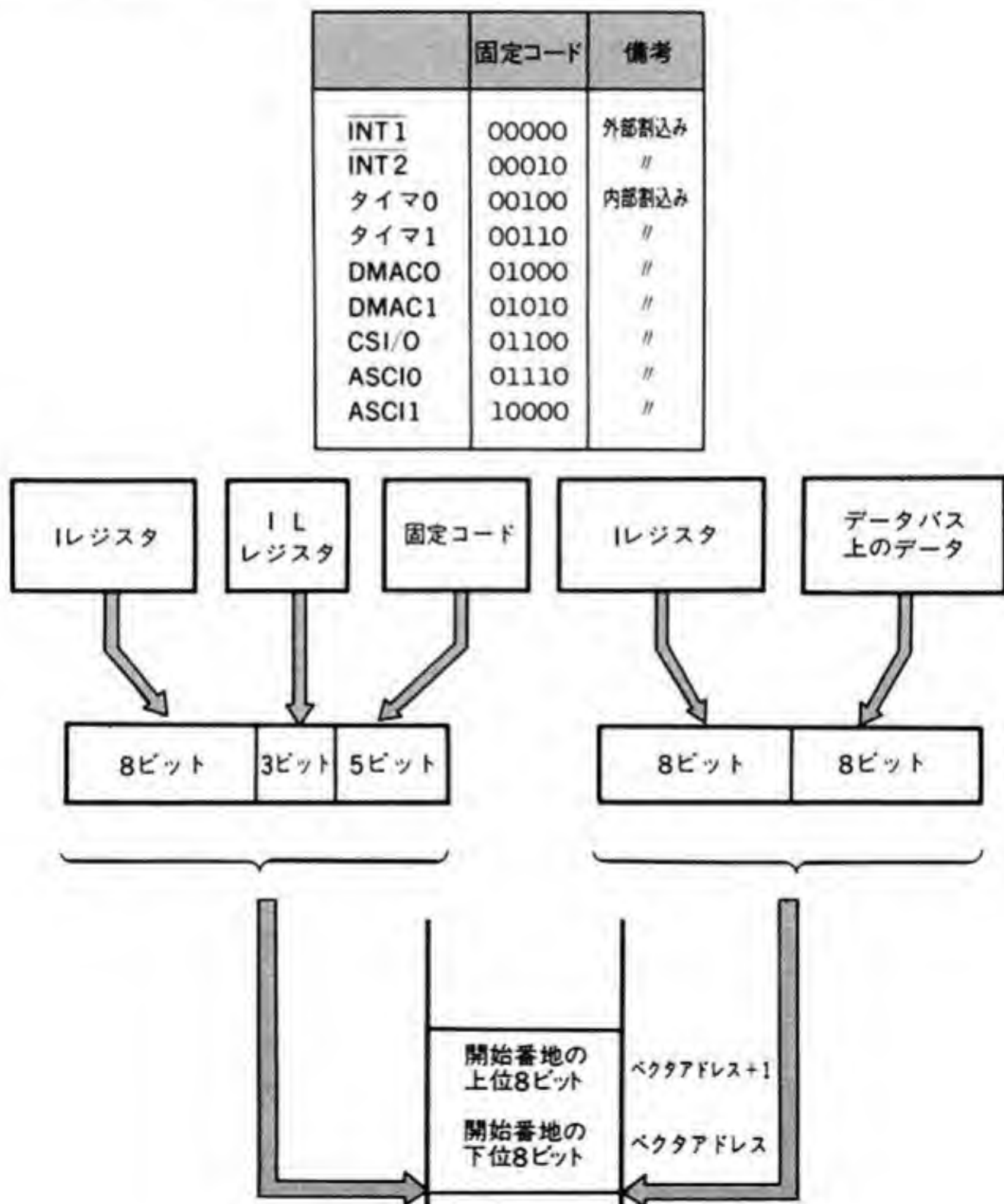


図 9・3 64180 のベクタ方式の割込み

9. 割 込 み

$\overline{\text{INT0}}$ モード 2 の場合、ベクタを生成するため外部にハードウェアが必要となります。Z80 の周辺 LSI を使用すればこの機能はサポートされていますが、68 系の周辺 LSI やインテルの 80 系周辺 LSI を使用する場合は、外付回路で対応することになります。これに対し、 $\overline{\text{INT1}}$ 、 $\overline{\text{INT2}}$ などのベクタアドレスは自動的に生成します。したがって、68 系やインテルの 80 系の周辺 LSI を使用する場合は、 $\overline{\text{INT1}}$ 、 $\overline{\text{INT2}}$ を使用するのが便利です。

IEF フラグ

IEF フラグは、直接 CPU からリードすることが可能です。LD A, I あるいは LD A, R 命令を実行することにより、IEF₂ の内容が P/V フラグに転送され、IEF フラグの内容を知ることができます。ここで注意すべき点は、Z80 では LD A, I あるいは LD A, R 命令の実行中に割込みが発生した場合、割込み発生後の IEF フラグが P/V フラグに反映され、正しい値が読み出せなくなります。そこで、64180 (R1/Z) では LD A, I と LD A, R 命令の最後では、割込みをサンプリングしません。したがって、これらの命令を実行すると、IEF フラグの正しい値が P/V フラグに反映されます。

9・3 マスカブルな割込み

〔1〕 **IEF フラグ** 64180 はソフトウェアで割込み禁止可能なマスカブル割込みとして、内部 7 本、外部 3 本の要因があります。これらのマスカブルな割込みは、**IEF (Interrupt Enable Flag) フラグ**により制御されています。IEF フラグはリセット直後 '0' に初期化されており、この状態ではマスカブルな割込みを受け付けることはできません。IEF フラグを '1' にセットし、マスカブルな割込みを受け付け可能とするには **EI 命令** を実行する必要があります。また、IEF フラグは **DI 命令** により '0' にリセットすることも可能です。また、マスカブルな割込みが受け付けられると IEF フラグが '0' にリセットされ、他の割込みが受け付け不可能となります。再度割込み受け付け可能とするためには、**EI 命令** を実行します。一般に、EI 命令は割込み処理ルーチンの終了する **RETI 命令** の直前に実行します。

〔2〕 **\overline{INT}** 64180 は、外部入力の 3 本の割込み要求端子をもっています。この 3 本の \overline{INT} 端子 ($\overline{INT0}$, $\overline{INT1}$, $\overline{INT2}$) は、IEF 以外に個別に割込み許可フラグをもっています。このフラグは、I/O 空間上に配置された **INT/TRAP** コントロールレジスタに配置されています。このフラグはそれぞれ **ITE0**, **ITE1**, **ITE2** と呼ばれ、リセット後 '100' に初期化されます。したがって、 $\overline{INT0}$ のみ **EI 命令** 実行後すぐに受け付け可能であり、 $\overline{INT1}$, $\overline{INT2}$ はマスクされた状態になっています。 \overline{INT} 端子はレベルセンスの割込みです。これらの割込み要求は、各命令の最後のマシンサイクルの後ろから 2 番目の ϕ クロックの立下りエッジ

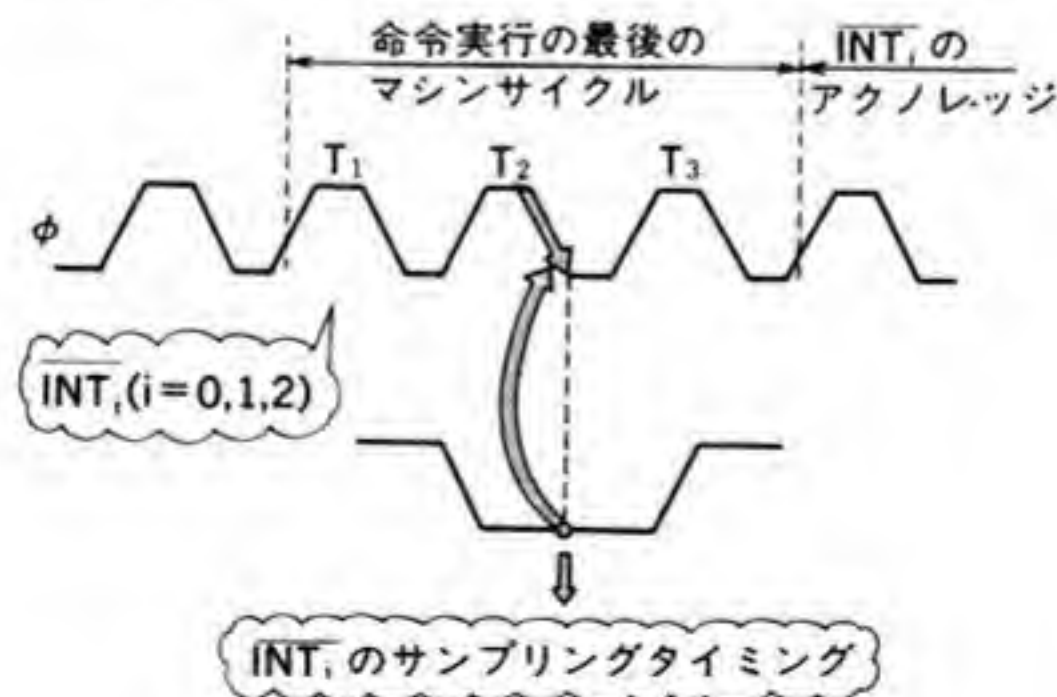


図 9・4 \overline{INT} のサンプリング

9. 割 込 み

ジでサンプリングされています。サンプリングのタイミングを図 9・4 に示します。

〔3〕 内部 割 込 み 内蔵の周辺からの割込み要因も、IEF フラグ以外に要因別の割込み許可フラグをもっています。これらの割込み許可フラグはリセット直後 '0' に初期化されています。したがって内部割込みを発生させるには、EI 命令を実行するだけでなく、割込み要因ごとにもっている割込み許可フラグを '1' にセットしてやる必要があります。この割込み許可フラグは、I/O 空間上の各コントロールレジスタに配置されています。これらのレジスタとフラグ名を表 9・2 に示します。

表 9・2 内部割込みの割込み許可フラグ

周 辺	レ ジ ス タ		I/O アドレス	割込み許可 フラグ	リセット 直後の値
タイマ0	タイマコントロールレジスタ		10H	TIE0	0
タイマ1	タイマコントロールレジスタ		10H	TIE1	0
DMAC0	DMA ステータスレジスタ		30H	DIE0	0
DMAC1	DMA ステータスレジスタ		30H	DIE1	0
CSI/O	CSI/O コントロールレジスタ		0AH	EIE	0
ASCI0	送信	ASCIステータスレジスタ ch0	00H	TIE(0)	0
	受信	ASCIステータスレジスタ ch0	00H	RIE(0)	0
ASCI1	送信	ASCIステータスレジスタ ch1	01H	TIE(0)	0
	受信	ASCIステータスレジスタ ch1	01H	RIE(0)	0

9・4 $\overline{\text{INT0}}$ モード 0

$\overline{\text{INT0}}$ は、Z80 と同様にモード 0 からモード 2 の 3 つの動作モードをもっています。このうちモード 0 は、インテル社の 8080、8085 などがもっている割込みと同一の動作モードです。このモードを使用すると、たとえばインテル社の 8259 のような割込みコントローラを使用できます。

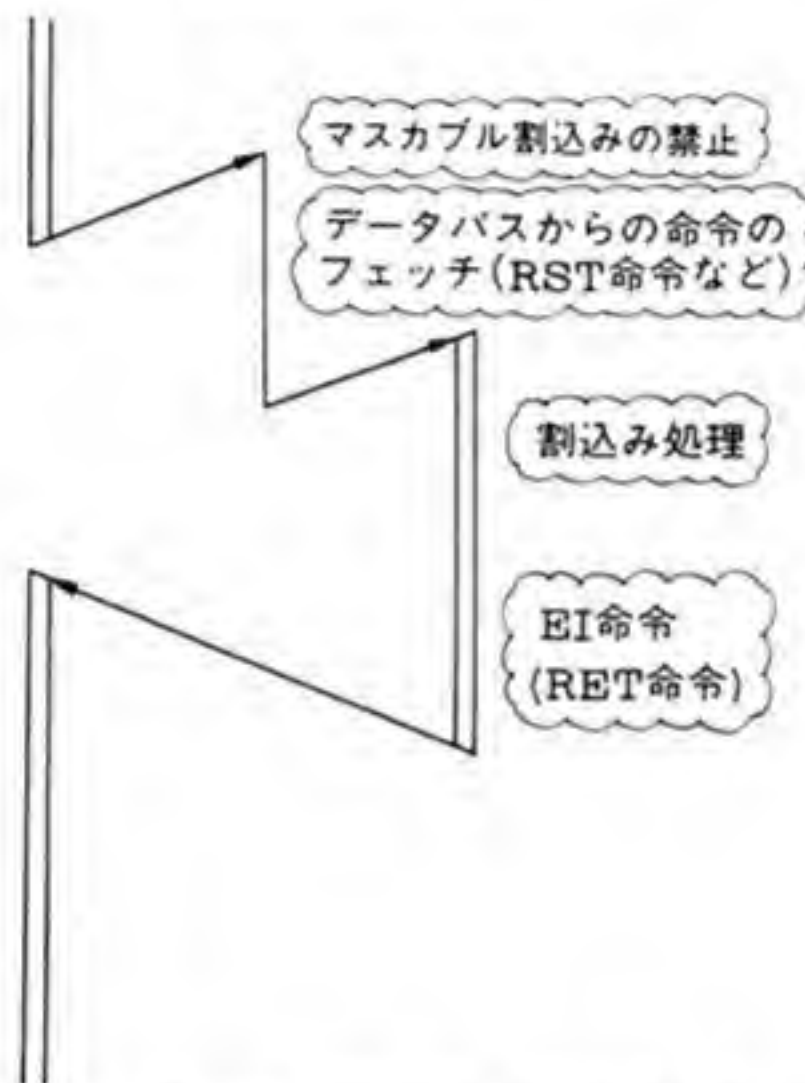


図 9・5 $\overline{\text{INT0}}$ モード 0 のシーケンス

このモードでは、割込みアクノレッジサイクル時にデータバス上のデータを命令としてフェッチし実行します。この命令としては、RST 命令や CALL 命令を使用し、そのジャンプ先に割込み処理のためのプログラムを配置します (図 9・5)。

割込み要求 ($\overline{\text{INT0}}$ 端子) がアクティブ (“Low” 入力) になると、CPU は割込みアクノレッジサイクルを実行します。このアクノレッジサイクル中には、 $\overline{\text{LIR}}$ 信号と $\overline{\text{IOE}}$ 信号がアクティブになります。8259 などの割込みコントローラを使用する場合、割込み要求に対してアクノレッジサイクル信号 ($\overline{\text{ACK}}$ 信号) を返す必要があります。この $\overline{\text{ACK}}$ 信号を作るために、 $\overline{\text{LIR}}$ 信号と $\overline{\text{IOE}}$ 信号を使用することができます。この $\overline{\text{ACK}}$ 信号を受けると、8259 はデータバス上に CALL 命令をのせてきます。64180 は、割込みアクノレッジサイクルの T_3 ステートの立上りで、CALL 命令のオペコードを取り込み実行します。図 9・6 と図 9・7 に $\overline{\text{INT0}}$ モード 0

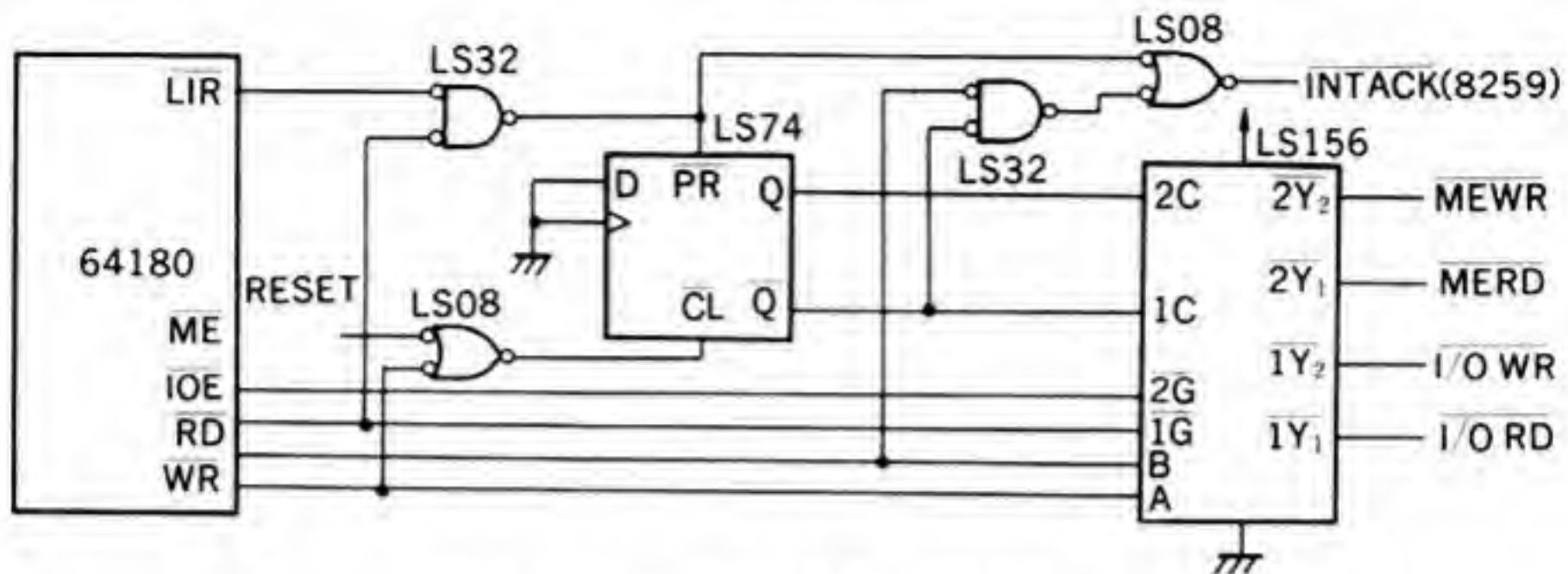


図 9・7 64180 による INTACK 作成例

のタイミングチャートと 8259 とのインタフェース例を示します。

CALL 命令のオペコードフェッチ時にはプログラムカウンタ (PC) は停止していますが、Z80 とは異なり 2 バイトのオペランドリード時にはプログラムカウンタが '+2' されています。このため、スタックに退避されるプログラムカウンタの値が実際に戻るべき番地より '+2' されています。したがって、割り込み処理ルーチン上でスタックの値を '-2' する必要があります。

なお、 $\overline{\text{INT0}}$ をモード 0 で使用するためには、IMO 命令で割り込みの動作モードをモード 0 に設定します。またリセット直後は、モード 0 にイニシャライズされます。

9.5 INT0モード 1

このモードでは、プログラムは固定番地 (0038H 番地) からスタートします。モード 1 では、モード 0 やモード 2 のように命令やベクタをデータバスにのせる必要がありません。このため、ソフトウェア、ハードウェアとも最も単純なもので済ませることができます。ところが、この番地は、RST_L38H 命令とジャンプ先の番地が重なっているため、もし RST 命令 (RST_L38H 命令) を使用する場合は注意が必要です。また複数の割込み要因がある場合、モード 0、モード 2 のように拡張性がないという不便さもあります。したがって、小規模なシステムでは大変に便利なモードですが、システム規模が大きく、外部割込み要因が多い場合はモード 0 かモード 2 を使う必要があります (図 9・8)。

INT0 をモード 1 で使用するためには、IM1 命令を実行しモード 1 を設定する必要があります。

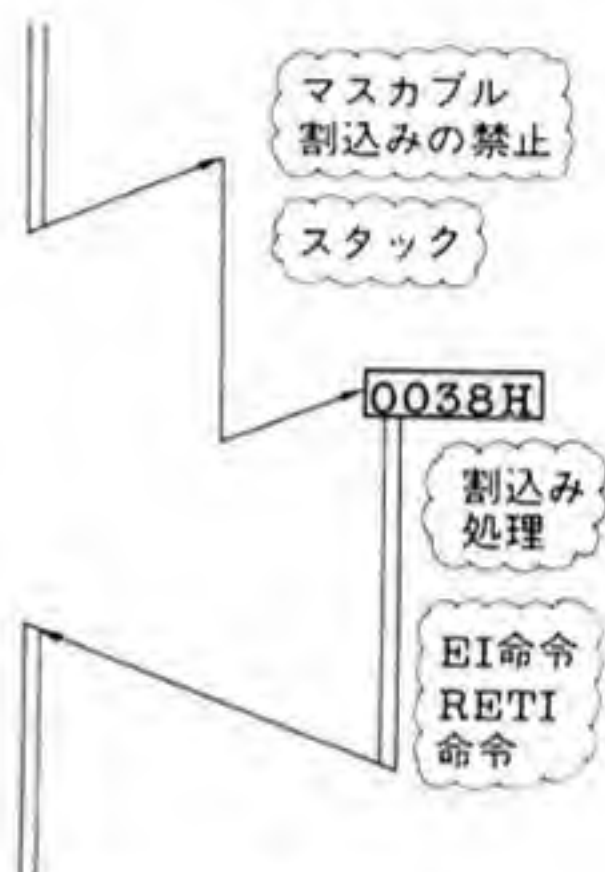


図 9・8 INT0モード1のシーケンス

EI 命令と RETI 命令

マスカブルな割込み (INT0 など) は、各命令の最後のマシンサイクル中にサンプリングされます。ただし EI 命令の実行中には、割込みは受け付けられません。したがって EI 命令を使用した場合、EI 命令の次の命令から割込みが受け付けられます。

通常 EI 命令は、割込み処理の最後に RETI 命令とペアで使用されます。したがって、EI 命令、RETI 命令の順番にプログラムを配置すると RETI 命令は必ず実行され、割込みの要求があった場合は、RETI 命令終了後に割込みアクノレッジサイクルが実行されます。

9・6 $\overline{\text{INT0}}$ モード 2

〔1〕 **モード 2 の機能** このモードは、サイログ社の Z80 周辺 LSI などがサポートしているベクタ方式の割込みです。このモードでは、割込みアクノレッジサイクル中にデータバスから下位側 8 ビットのベクタアドレスを読み込みます。このベクタアドレスと I レジスタ（上位側 8 ビットのベクタアドレスとして使用される）の値をあわせて、16 ビットのベクタアドレスを生成します。このベクタアドレスから、実際の割込み処理の開始アドレスをリードします。割込み処理は、ここで読み込んだアドレスから開始されます。

したがって、周辺ごとにこの下位側 8 ビットのアドレスを変更することにより複数（ $2^8/2 = 128$ 通り）の割込み要因を区別し、独立に処理をさせることができます。

$\overline{\text{INT0}}$ をモード 2 で使用するためには、IM2 命令を実行させる必要があります（図 9・9）。

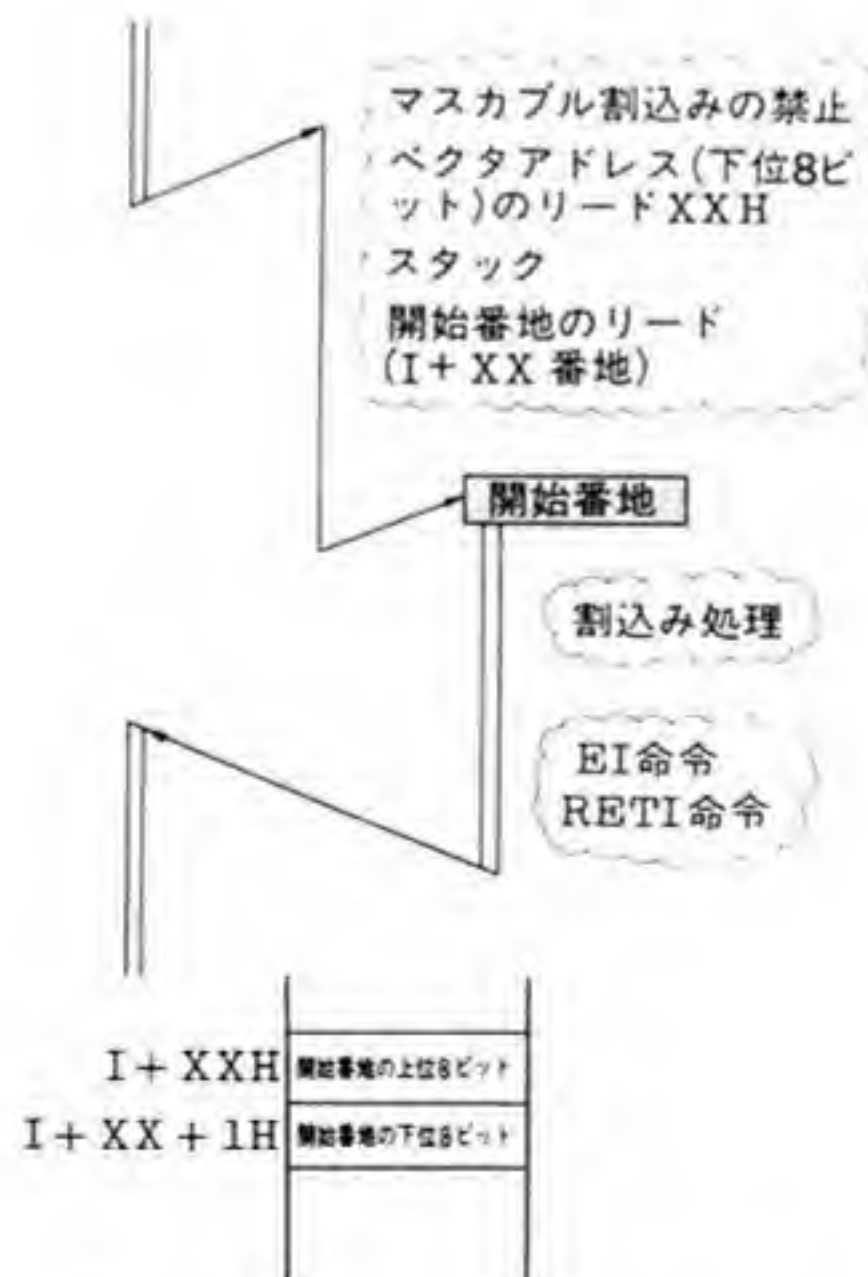


図 9・9 $\overline{\text{INT0}}$ モード 2 のシーケンス

9. 割 込 み

〔2〕 **デジチェーン** このモード2では、すでに述べたように複数の割込み要因をもつことができます。ところが、この状態では異なる要因から同時に要求が入った場合、優先度をつけることができません。そこで、優先順位をつける手法として Z80 周辺 LSI を使用した場合、**デジチェーン**と呼ばれる方法がとられます。デジチェーンはハード的に割込みの優先順位を決定する手法で、割込み許可信号 (IE 信号) により割込みの発生を許可するものです。図 9・10 にデジチェーンの構成例を示します。この IE 信号には入力信号 (IEI) と出力信号 (IEO) があります。IEI 信号は通常 'H' が入力され、'H' が入力されていれば IEO からは 'H' が出力されています。これが割込みの許可されている状態です。この状態で割込みが発生すると、その要因は IEO が 'L' となります。この IEI に 'H' が入り IEO に 'L' が出力されている状態が、割込み処理中の要因です。また、IEI に 'L' が入力されると IEO も 'L' 出力となります。これが割込み発生禁止された状態です。この機能により、デジチェーンの高位におかれた割込み要因に対し、優先的に割込み処理を行わせることができます。

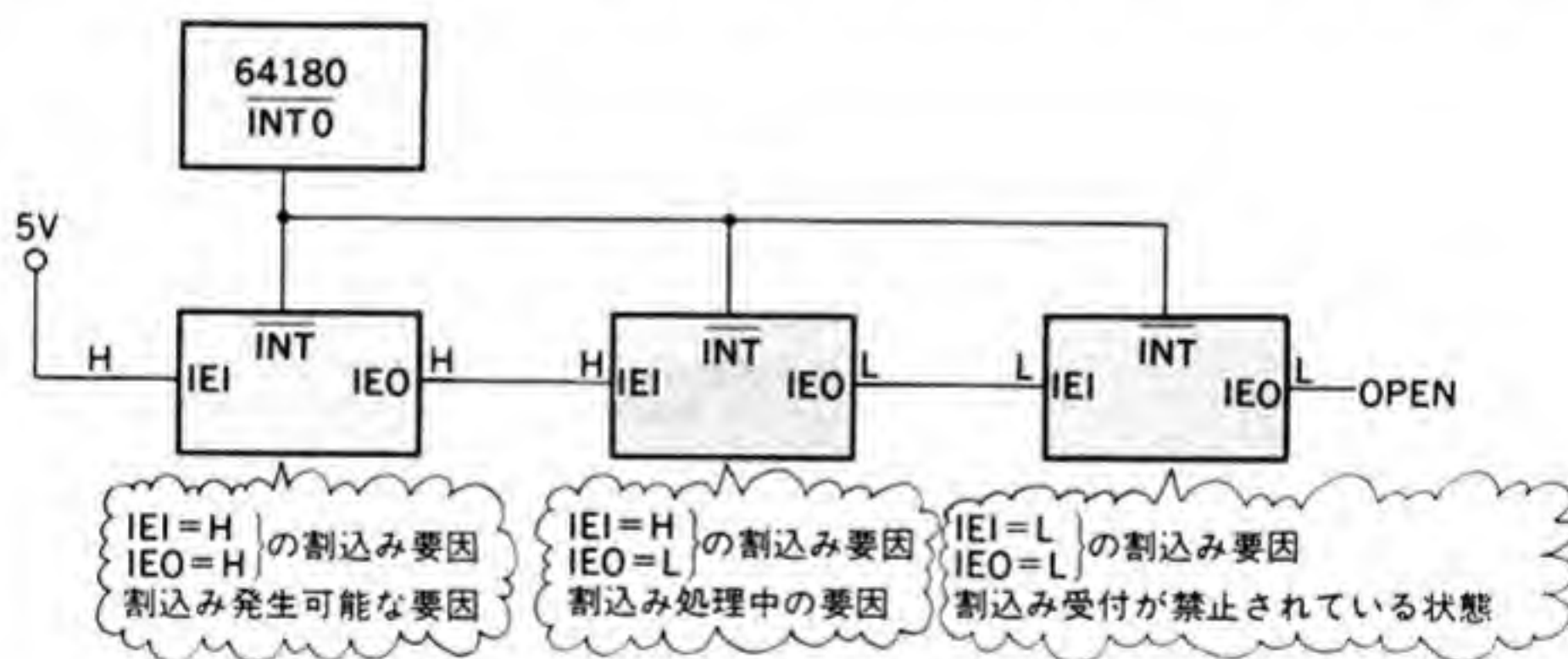
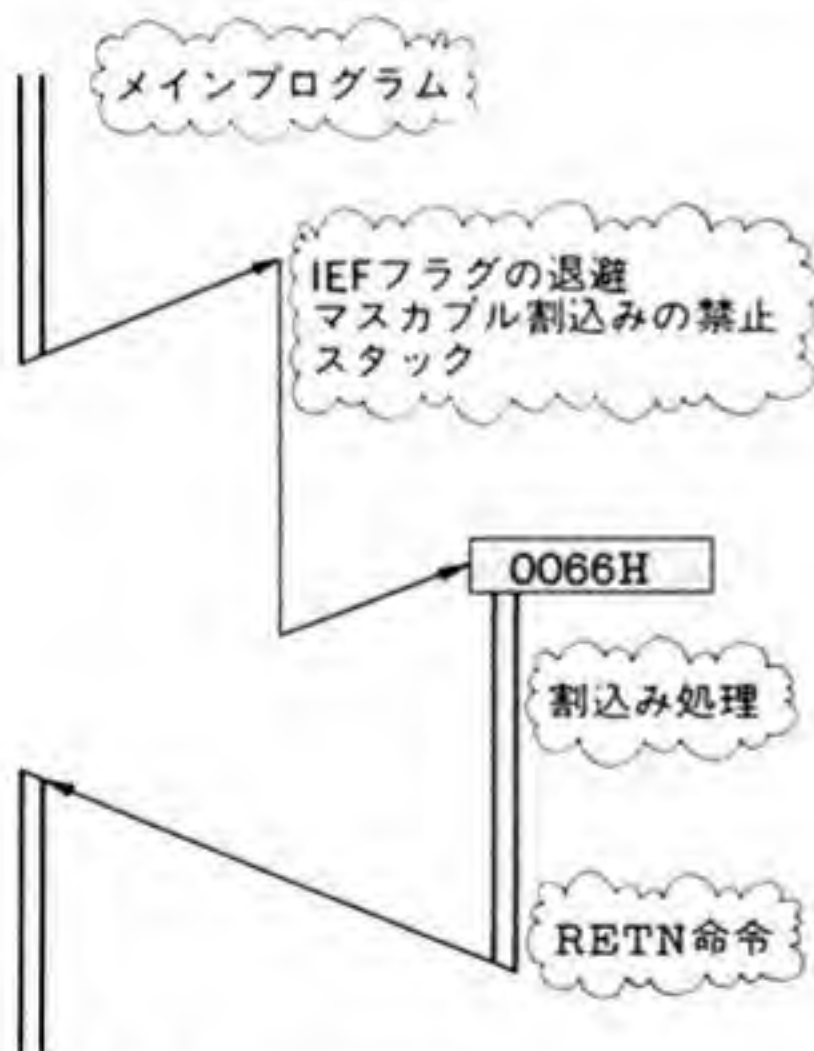
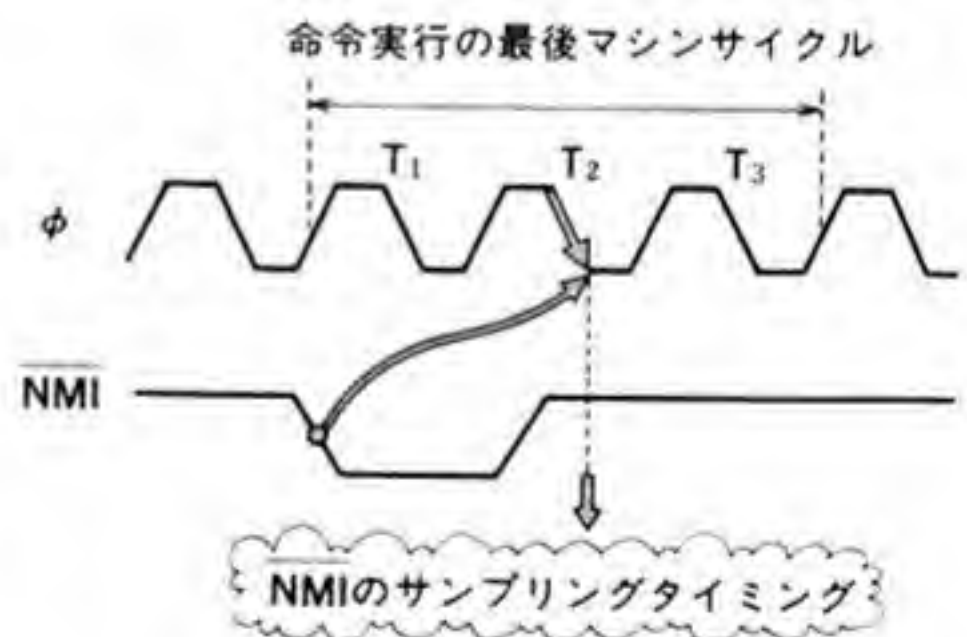


図 9・10 デジチェーン

9・7 ノンマスカブルな割込み

〔1〕 $\overline{\text{NMI}}$ $\overline{\text{NMI}}$ は、ソフトウェアでマスク不可能な割込みです。 $\overline{\text{NMI}}$ が発生すると、IEF フラグの値にかかわらず 66 番へのジャンプが発生します。 $\overline{\text{NMI}}$ は命令ごとにサンプリングされています。したがって $\overline{\text{NMI}}$ が入力されたとき、実行中の命令終了後、 $\overline{\text{NMI}}$ のアクノレッジサイクルに続き 66H 番地へのジャンプが発生します。 $\overline{\text{NMI}}$ は、各命令の最後のマシンサイクルの後から 2 番目の ϕ クロックの立下りエッジでサンプリングされています。このタイミングまでに $\overline{\text{NMI}}$ が検出されると、そのマシンサイクル終了後、64180 は $\overline{\text{NMI}}$ のアクノレッジサイクルを実行します。また、 $\overline{\text{NMI}}$ はエッジセンスであり、一度検出された $\overline{\text{NMI}}$ の立下りエッジは、サンプリングのタイミングまで内部に保持されています。この $\overline{\text{NMI}}$ 処理からの復帰には RETN 命令を使用します (図 9・11、図 9・12)。

図 9・11 $\overline{\text{NMI}}$ のシーケンサ図 9・12 $\overline{\text{NMI}}$ のサンプリング

〔2〕 $\overline{\text{NMI}}$ の用途 通常のプログラム実行に $\overline{\text{NMI}}$ が用いられることは少なく、停電などによる電源の遮断などの緊急の用途に使用するのが通例です。また 64180 の場合、 $\overline{\text{NMI}}$ が内蔵 DMAC を一時停止する機能があります。たとえば、DMAC がメモリ間のバースト転送を行っている場合、外部 I/O の処理を優先的に行いたい場合、 $\overline{\text{NMI}}$ を用いて DMAC を停止させることができます。

9. 割 込 み

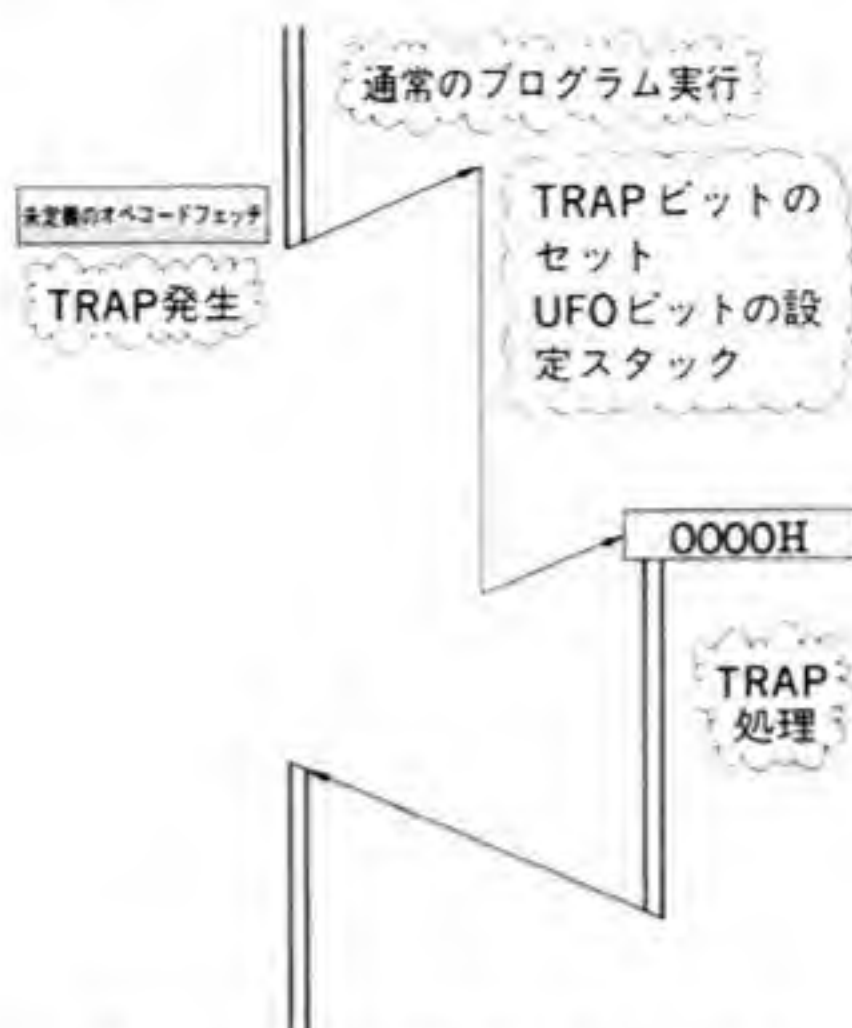


図 9・13 TRAP 割込みのシーケンス

〔3〕 **TRAP** TRAP はシステムのフェールセーフのための機能です。CPU が未定義のオペコードをフェッチすると割込みが発生します。すなわち、データバス上にのったノイズによるデータの変化や暴走が発生した場合に、システムのリスタートを図るための機能です。未定義コードは、2 バイト、3 バイト目に存在します (図 9・13)。

〔4〕 **TRAP の機能** TRAP が発生すると、リセットと同じ 0 番地へのジャンプが発生します。リセットルーチン上で、I/O レジスタにある TRAP ビット (INT/TRAP コントロールレジスタのビット 7) をテストすることにより、0 番地からの実行がリセットか TRAP かを判定することができます。また、同レジスタビットにある UFO ビットを判定することにより、TRAP が 2 バイト目のオペコードで発生したか 3 バイト目のオペコードで発生したかを判定することができます。TRAP はソフトでマスク不可能であるため、TRAP 処理中に再度 TRAP が発生する場合があります。

9・8 RET命令, RETI命令, RETN命令

〔1〕 割込みルーチンからの復帰 マスカブル割込みの割込み処理ルーチンからメインルーチンに復帰する場合、通常 RETI 命令や RET 命令を使用します。この2つの命令の動作は全く同一です。ただし、Z80系の周辺LSIを使用する場合は、RETI 命令を使用します。これは、Z80 周辺 LSI が割込み要求フラグを RETI 命令を用いてリセットする（実際には RETI 命令のオペコードをデコードする）ためです（表 9・3）。

表 9・3 RET命令, RETI命令, RETN命令

命 令	機 能	用 途
RET 命令	スタックの回復 (SP)→PC _L	内部割込み処理ルーチンからの回復 回復サブルーチンからの回復
RETI 命令	(SP+1)→PC _H SP+2→SP	Z80 周辺 LSI の割込み処理ルーチンからの回復
RETN 命令	スタックの回復 (SP)→PC _L (SP+1)→PC _H SP+2→SP IEF フラグの回復 IEF ₂ →IEF ₁	$\overline{\text{NMI}}$ 処理ルーチンからの回復

〔2〕 内部割込みと Z80 周辺 LSI 内部割込みと Z80 周辺 LSI を同時に使用する場合は、RET 命令と RETI 命令を使い分ける必要があります。たとえば、内蔵の周辺 I/O から割込み処理の終了時に RETI 命令を使用すると、割込み処理が行われていないにもかかわらず外部の Z80 周辺 LSI の割込み要求がリセットされてしまう可能性があります。これを避けるため内蔵の周辺 I/O と Z80 周辺 LSI を同時に使用する場合、内蔵の周辺 I/O の割込み処理ルーチンには RET 命令を、Z80 周辺 LSI の処理ルーチンには RETI 命令を使用します。

〔3〕 RETN 命令と IEF フラグ $\overline{\text{NMI}}$ の処理ルーチンからの復帰には RETN 命令を使用します。RET 命令や RETI 命令との大きな違いは IEF フラグのスタック動作です。 $\overline{\text{NMI}}$ では、IEF₁ フラグの内容が IEF₂ フラグに退避されます。このとき IEF₁ フラグは 0 にリセットされ、マスカブルな割込みは受け付けられま

9. 割 込 み

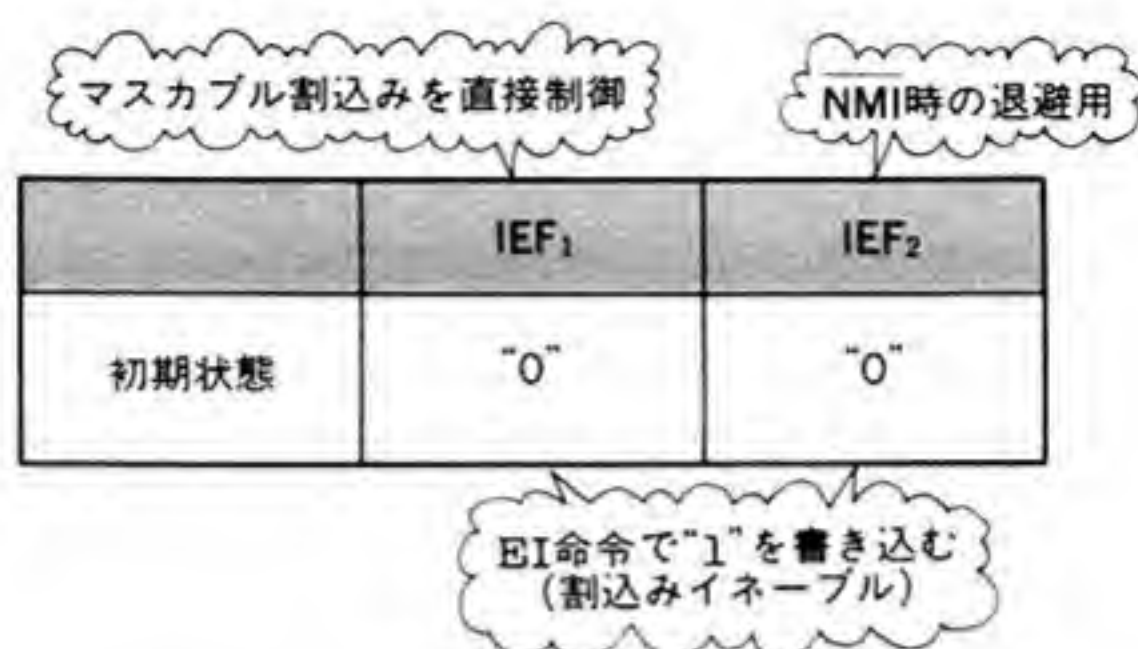


図 9・14 IEF フラグ

せん。RETN 命令ではこれを復帰し、IEF₂の内容を IEF₁ フラグに転送します。これに対し、RET 命令や RETI 命令を実行しても、IEF フラグにはなんの影響も及ぼしません (図 9・14)。

このように、内蔵の周辺の割込みには RET 命令を、Z80 周辺 LSI の割込みには RETI 命令を、NMI には RETN 命令を使用します。

10. 特殊動作モード

64180 は、ホルト、バスリリース、低消費電力の3つの特殊動作モードをもっています。ホルトモードは、CPU が処理すべきタスクがなくなり割込み待ちとなるモードです。バスリリースモードは、外部のバスマスタがバスを使用できるように、CPU がバスを解放するモードです。低消費電力モードは、CPU、周辺機能が停止し、消費電力の低減を図るモードです。また、64180 を初期化するためリセットの機能があります。

10・1 リセット

パワーオン時、あるいは動作中のシステムを初期化するため、リセットを行います。64180 は、リセット端子を 6 ステート以上アクティブ（Low 状態）にすることによりリセット状態になります。リセットにより、CPU と周辺機能は以下に述べるとおりに初期化されます。

〔1〕CPU 64180 は、リセットによりその動作を中止します。リセット信号がアクティブ（Low 状態）の間、アドレスバスとデータバスはハイインピーダンス状態になります。ME、IOE などの制御信号は、インアクティブな状態（'High'）に固定されます（図 10・1）。

リセット信号がインアクティブな状態（'High'）になると、CPU は論理アドレスの 0000H 番地から実行を開始します。リセット直後、MMU のベースレジスタは 00H に初期化されます。このため、見かけ上 MMU の動作がバイパスされ、Z80 と同様な 64 K バイトのアドレス空間をもつ CPU に見えます。このとき物理アドレス上では、00000H 番地からの実行となります。

	リセット中の状態	
$\overline{\text{LIR}}$	1	
$\overline{\text{ME}}$	1	
$\overline{\text{IOE}}$	1	
$\overline{\text{RD}}$	1	
$\overline{\text{WR}}$	1	
$\overline{\text{REF}}$	1	
$\overline{\text{HALT}}$	1	
$\overline{\text{BUSACK}}$	1	
ST	1	
アドレスバス	ハイインピーダンス	
データバス	ハイインピーダンス	入力状態

図 10・1 リセット中の制御信号

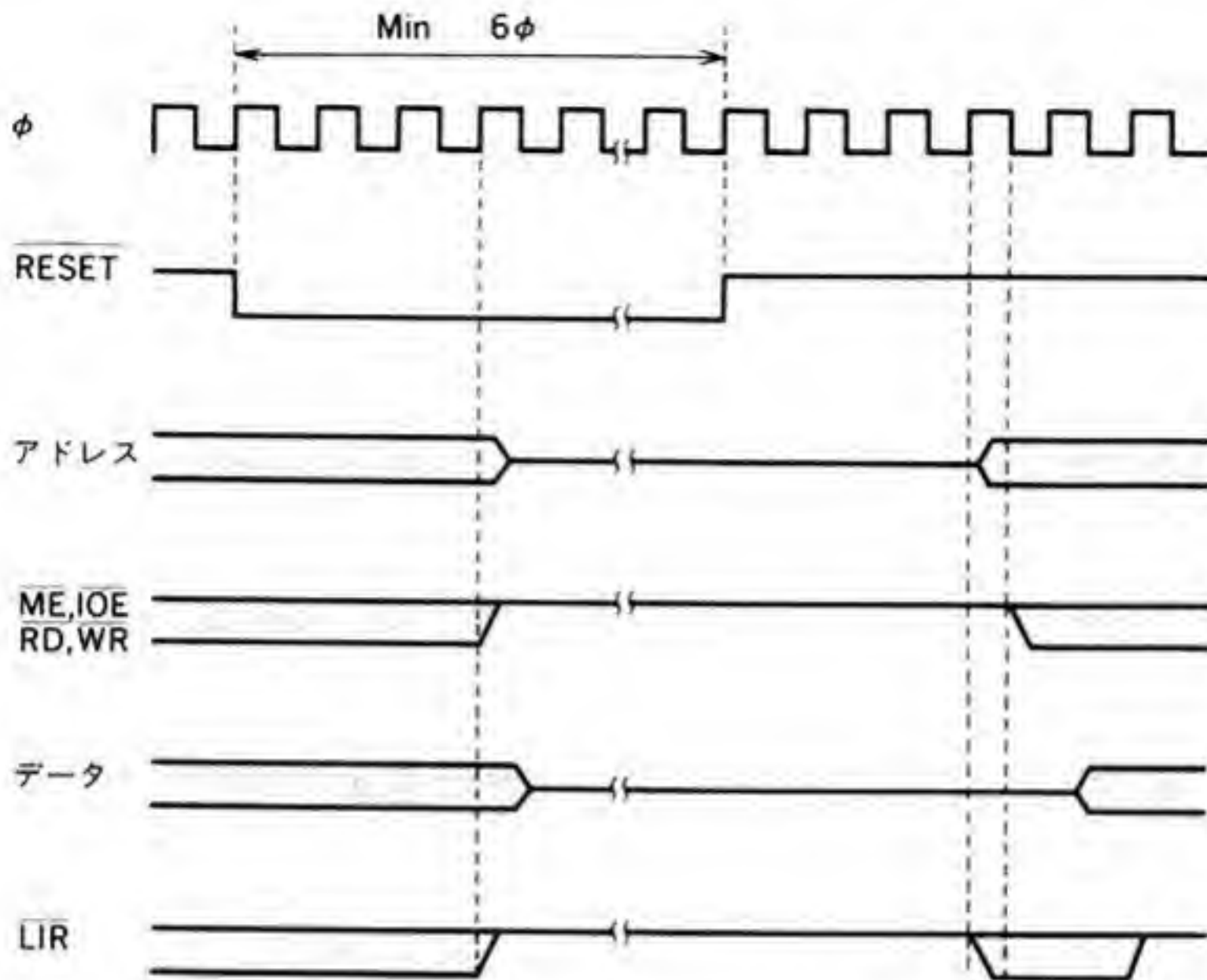


図 10・2 リセットのシーケンス

CPU のレジスタのうち、アキュムレータ、汎用レジスタ (B, C, D, E, H, L)、インデックスレジスタ (IX, IY) は初期化されません。これに対して、スタックポインタ (SP)、I レジスタ、リフレッシュカウンタ ('R') は、'0' に初期化されます (図 10・2)。

また、割込みは IEF フラグが '0' に初期化されるため、マスカブルな割込みは禁止されています。マスカブルな割込みを使用可能とするには、EI 命令を実行する必要があります。

〔2〕周 辺 機 能 リセット直後、64180 の内蔵する DMAC、タイマ、CSI/O、ASCI の動作はディスエーブルされています。これらの周辺機能は、動作を許可するコントロールビットを I/O 空間に配置されたコントロールレジスタ上にもっており、リセット後、これらのビットはすべてディスエーブルされています。リセット直後の各コントロールレジスタの状態は付録を参照して下さい。

10・2 ホルトモード

ホルトモードは、CPUがHALT命令（76H）を実行した状態です。このモードでは、HALT命令の次のアドレスに配置された命令を繰り返しフェッチし、プログラムの実行が停止した状態になっています。プログラムカウンタも変化しません。ホルトモード中は、 $\overline{\text{HALT}}$ 信号が‘L’となりホルトモードであることを

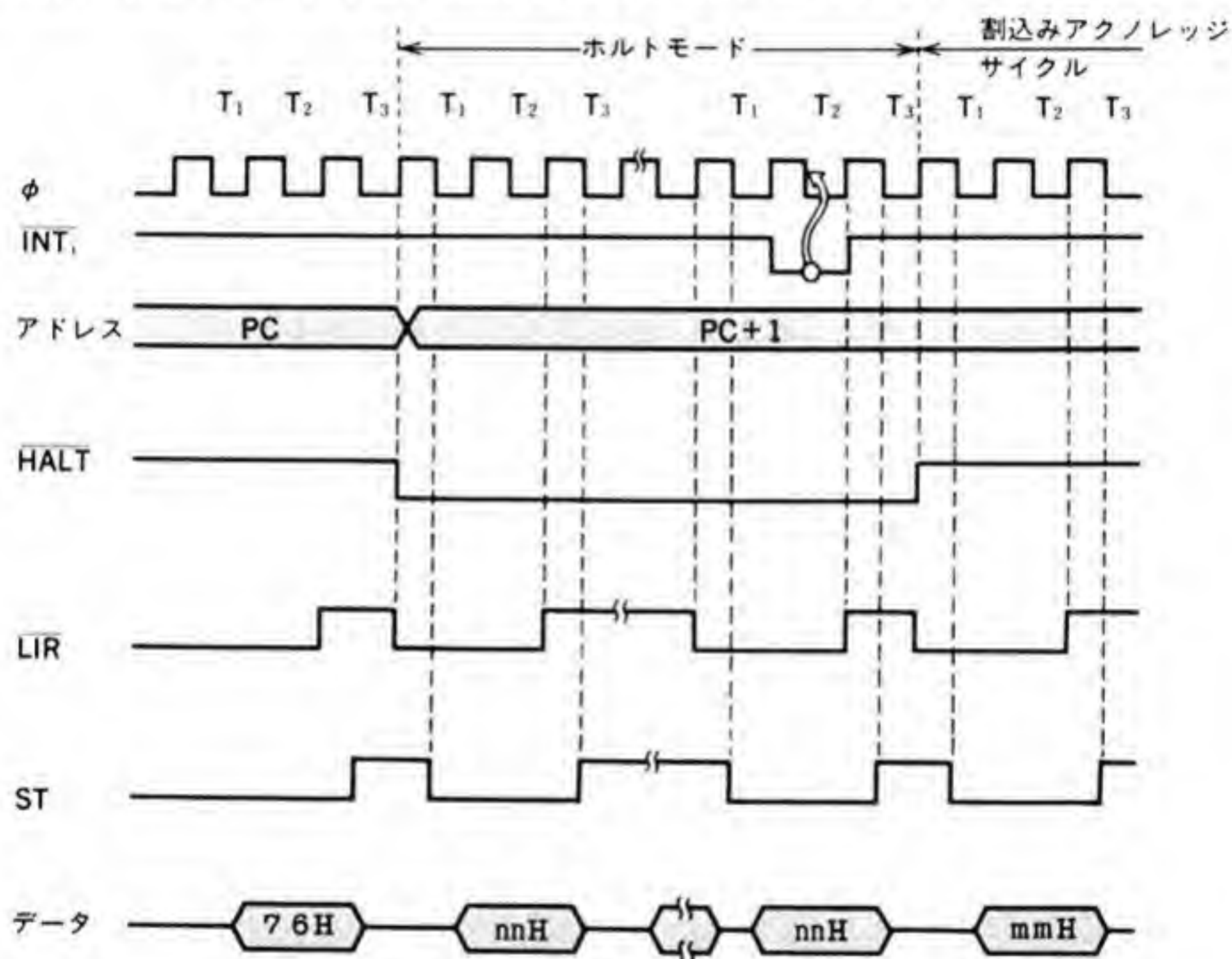


図 10・3 ホルトモード

表 10・1 ホルトモード中の各機能の動作状態

機 能	状 態	備 考
CPU	停 止	割込み、BUSREQ は受付可
リフレッシュ	継続動作可能	—
DMAC	継続動作可能	—
CSI/O	継続動作可能	—
ASCI	継続動作可能	—
タイマ	継続動作可能	—
CPG	継続動作	—

知らせます。また、命令を繰り返しフェッチしていますから、 $\overline{\text{LIR}}$ 信号が繰り返し出力されます (図 10・3)。

ホルトモード中でも、リフレッシュコントローラや DMAC、および他の周辺機能 (タイマ、CSI/O、ASCI) は継続して動作します。また、割込みや $\overline{\text{BUSREQ}}$ 要求も受付可能です (表 10・1)。

ホルトモードから通常動作モードに復帰するためには、割込みかリセットを使用します。割込みの要因としては、内部、外部の要因を使用可能です。ただし、 $\overline{\text{NMI}}$ を除いた他のマスカブル割込みを使用する場合、IEF フラグが '1' にセットされていないと、ホルトモードは解除されません。

ホルトモードはシングルタスクの応用で、I/O デバイスの処理を実行中、他に処理すべきタスクがない場合、I/O の処理の終了をもつ場合などに使用します。

ホルトモードとスリープモード

ホルトモードもスリープモードも、機能的には CPU が停止し割込み待ちの状態になり、同一のものといえます。スリープモードの場合、ホルトモードが命令のフェッチを行っているのに対し、命令フェッチそのものも停止し低消費電力化を図っています。したがって、消費電力が問題となる場合はスリープモードを使用するとよいでしょう。一方、スリープモードの場合リフレッシュが行われなくなるため、DRAM を使用する場合はホルトモードを使用するとよいでしょう。

10・3 バスリリースモード

$\overline{\text{BUSREQ}}$ 端子をアクティブ (Low 状態) にすることにより、64180 はバスリリースモードに入ります。バスリリースモードは、マルチ CPU システムや外部の DMAC など、外部のバスマスタとなり得る LSI に対しバス権を与えるための機能です。 $\overline{\text{BUSREQ}}$ 信号は、通常動作モードの場合、 T_3 サイクルの 1 つ前の ϕ クロックの立下りでサンプリングされます。また、バスリリースモード中は毎サイクルサンプリングされており、サンプリングされたクロックの次の ϕ クロックでバスリリースモードが終了し、通常動作モードとなり 64180 の CPU または DMAC がバスマスタとなります。図 10・4 に $\overline{\text{BUSREQ}}$ 信号のサンプリングタイミングを示します。バスリリースモードに入ると、アドレスバス、データバス、 $\overline{\text{ME}}$ 、 $\overline{\text{IOE}}$ 、 $\overline{\text{RD}}$ 、 $\overline{\text{WR}}$ 信号がハイインピーダンス状態になり、外部のデバイスがバスマスタとなることが出来ます。また、バスリリースモード中は $\overline{\text{BUSACK}}$ 信号がアクティブ (Low 状態) となり、外部のバスマスタに対し 64180 のバスがハイインピーダンスとなり、システムバスを使用可能であることを知らせます。

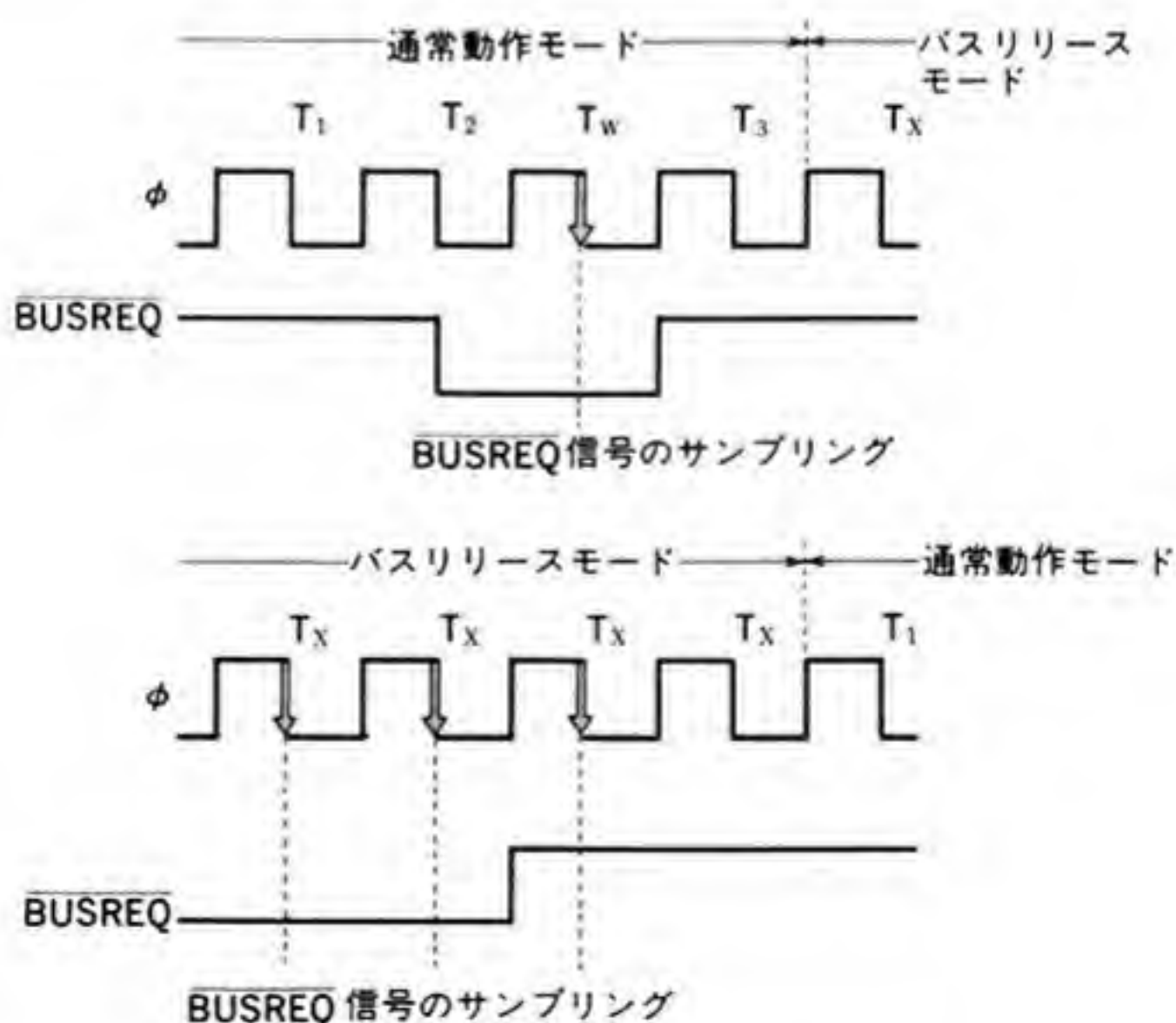


図 10・4 $\overline{\text{BUSREQ}}$ 信号のサンプリング

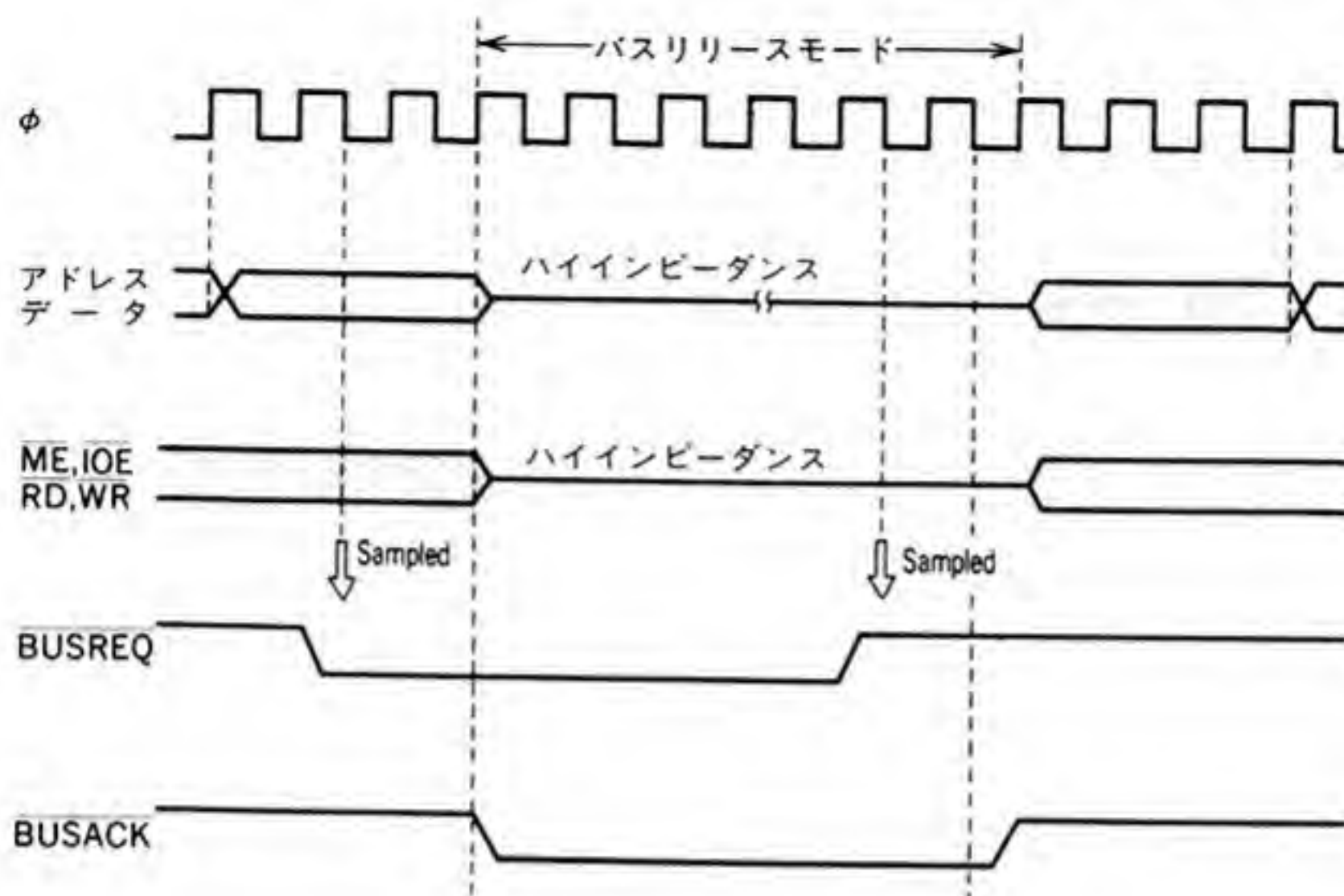


図 10・5 バスリリースモード

バスリリースモード中は割込みは受け付けられず、リフレッシュサイクルは挿入されません。したがってバスリリースモード中は、バスマスタとなった外部のデバイスが DRAM のリフレッシュを行う必要があります。このときでも、周辺機能（タイマ、CSI/O、ASCI）は継続して動作します（図 10・5）。

また、内蔵の DMA が動作中でも $\overline{\text{BUSREQ}}$ 信号は受け付けられます。このため、外部の DMAC が優先的にバスを使用することができます。

バスリリースモードから通常モードに戻るためには、前述のとおり、 $\overline{\text{BUSREQ}}$ 端子をインアクティブな状態（'High'）にします。この手順により、64180 の CPU または DMA がバス権を獲得し、 $\overline{\text{BUSACK}}$ 信号がインアクティブな状態（'High'）となります。

10・4 低消費電力モード

64180 は、低消費電力モードとして

スリープモード

システムストップモード

の2つのモードをもっています。これ以外に、システムストップモードへの遷移状態として、I/O ストップモードと呼ばれる状態があります。

表 10・2 低消費電力モード

	モードの設定	オペレーション		モードの解除
		CPU	周 辺	
スリープモード	SLP 命令の実行	停 止	動 作	RESET 割込み
システム ストップモード	IOSTP ビット (I/O コ ントロールレジスタ) をセットした状態での SLP 命令の実行	停 止	停 止	RESET 外部割込み

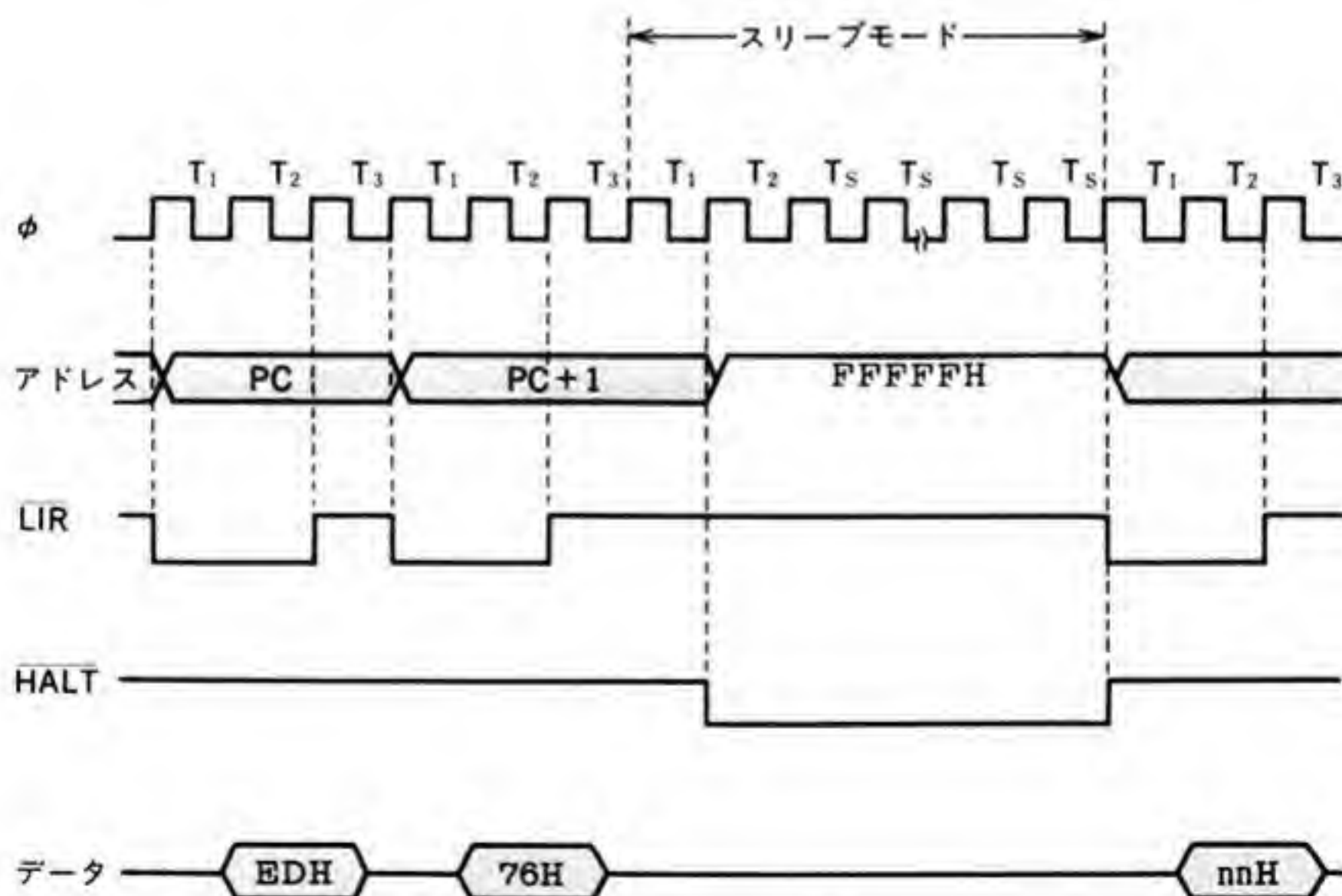


図 10・6 スリープモード

いずれのモードも、発振回路およびクロック出力回路は停止していません。したがって、Eクロックとφクロックは継続して出力されます(表10・2)。

(1) **スリープモード** CPUがSLP命令(ED 76H)を実行すると、64180はスリープモードに入ります。スリープモードに入ると、CPUは命令の実行を停止し、消費電力は通常動作時の60%程度になります。また、リフレッシュサイクルは挿入されず、DMACも動作しません。このとき、これ以外の周辺機能(タイマ、CSI/O、ASCI)は継続して動作します。BUSREQ信号は受け付けられ、外部のLSIがバスマスタになることができます。

スリープモードを解除するには、割込みかりセットを使用します。スリープモードの解除にマスカブルな割込みを使用する場合、IEFフラグが'1'にセットされているときは、通常の割込み処理ルーチンを実行します。これに対し、IEFフラグが'0'にリセットされている場合は、SLP命令の次の命令から実行が再開されます(図10・6)。

(2) **システムストップモード** I/Oコントロールレジスタ(I/O空間の3FH番地に配置)上のIOSTPビット(ビット5)を'1'にセットした状態で、SLP命令を実行することにより、64180はシステムストップモードに入ります。

このモードでは、CPU、周辺機能(DMAC、タイマ、CSI/O、ASCI)も停止した状態となり、消費電力は通常動作時の1/4程度になります。

システムストップモードを解除するためには、外部割込みとりセットを使用します。割込みによりCPUは動作を再開しますが、IOSTPビットは、'1'にセットされた状態で周辺機能は停止しています。周辺機能の動作を再開させるためには、IOSTPビットに'0'を書き込みます。

11. DMA コントローラ

64180 は、2 チャンネルの DMA コントローラを内蔵しています。DMAC は CPU に代わり、メモリ↔メモリ、メモリ↔I/O の高速なデータ転送を行うための周辺機能です。たとえば、フロッピーディスクコントローラやハードディスクコントローラとのデータ転送には欠かせない機能です。DMAC はマルチタスクを処理するためには必須の機能で、CPU は一度 DMAC に起動をかけた後は、データ転送をいっさい気にせず、プログラムの処理を行うことができます。

11・1 DMAC とは

データをI/Oデバイスからメモリに転送したり、あるメモリエリアから他のメモリエリアに転送したりする手法には、大きく分けて次のような3通りの方法があります（図11・1、表11・1）。

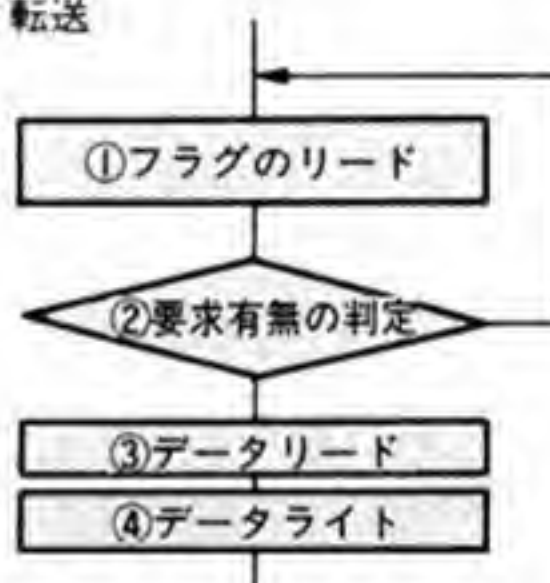
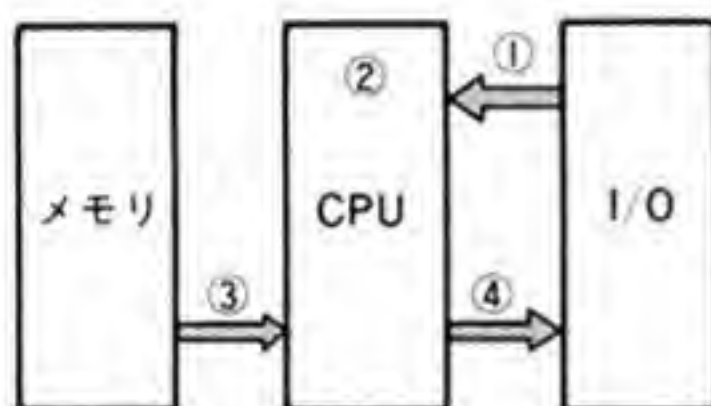
(1) CPU（プログラム）によるデータ転送…ポーリング

(2) 割り込みによるデータ転送

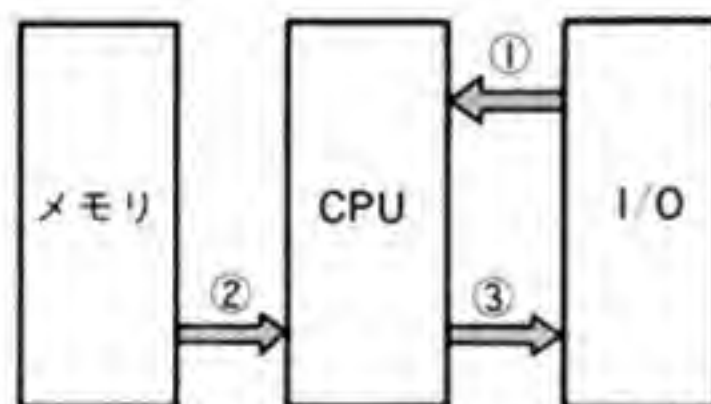
(3) DMACによるデータ転送

(1)、(2)はいずれも、データ転送はCPUが行います。(1)は転送すべきデータ

(1) CPU(プログラム)によるデータ転送



(2) 割り込みによるデータ転送



(3) DMACによるデータ転送

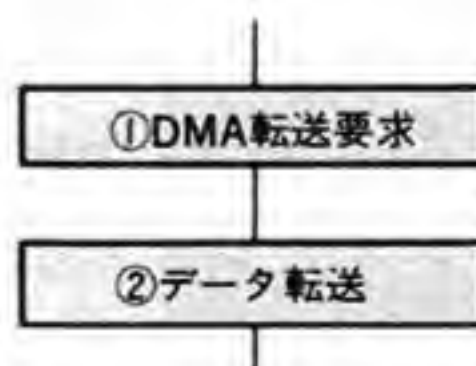
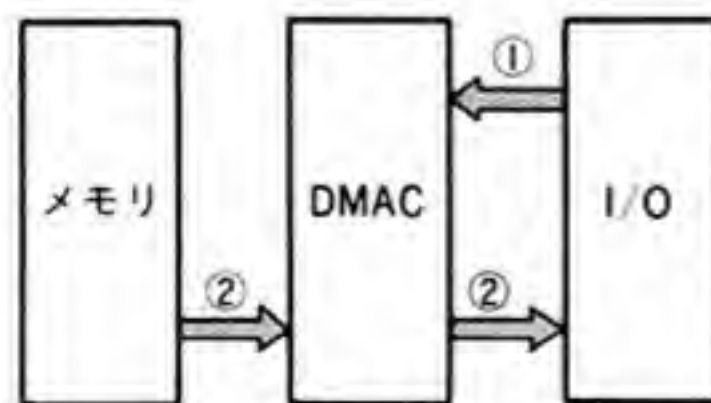


図 11・1 データ転送の手法

表 11・1 データ転送の手法の比較

	長 所	短 所
プログラムによる 転送(ポーリング)	一番手軽なデータ転送	外部との同期がとりにくい プログラム処理速度が遅い
割 込 み	外部との同期がとりやすい	アクノレッジサイクルによる速度の低下 割込みのハードが必要
D M A C	転送速度が速い 外部との同期がとりやすい	DMA の専用ハードが必要

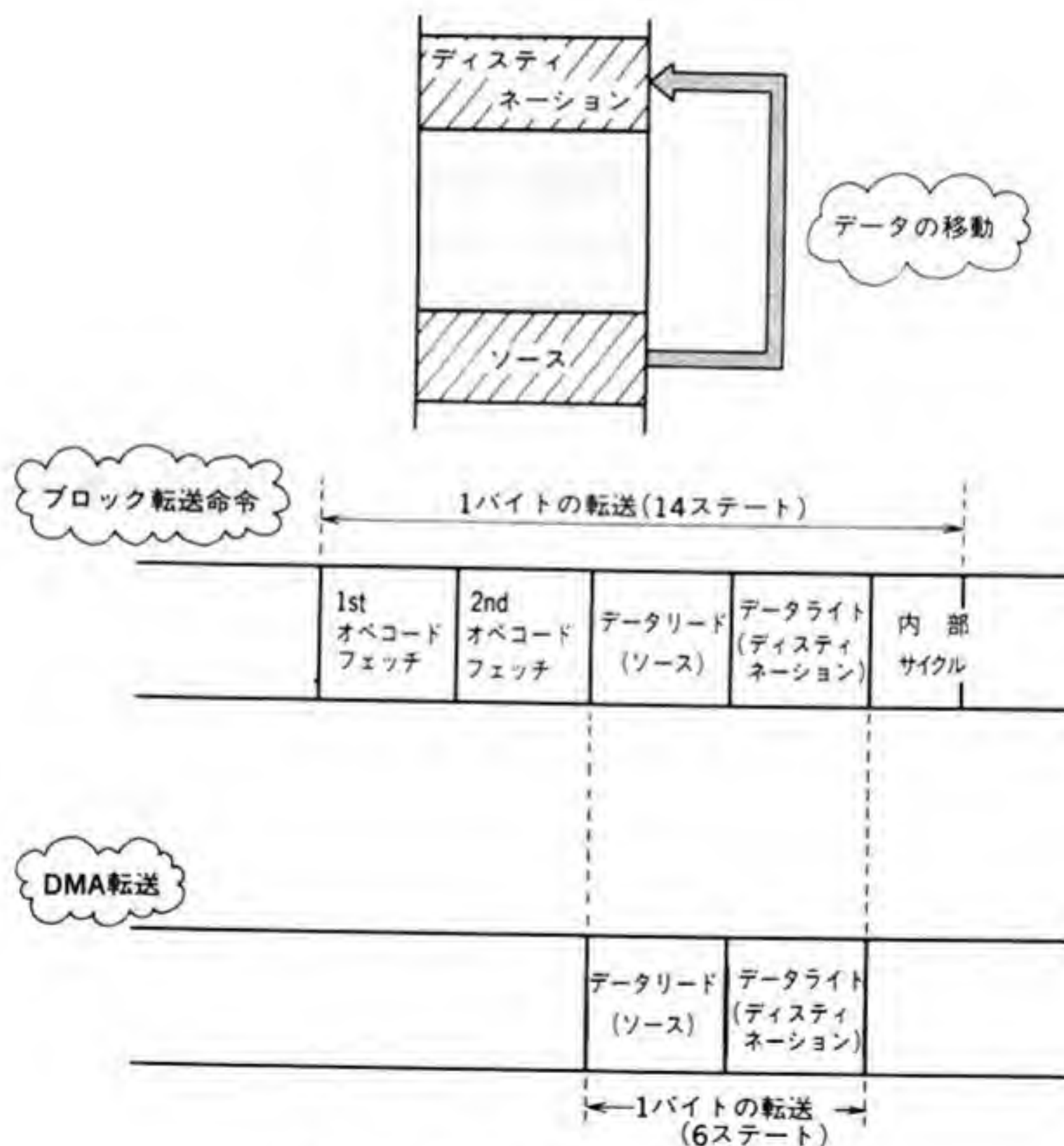


図 11・2 ブロック転送命令と DMA 転送

11. DMA コントローラ

の有無を CPU がチェックし、命令でデータのリードおよびライトを独立に行う必要があります。また、(1)の手法では、データの有無の判定およびデータ転送を行う間、CPU が他のプログラムを実行できないため、システムのスループットが下がるという欠点があります。

(2)の手法では、データの有無を判定するためのプログラムが、割込みにより省略可能となります。しかし、(1)、(2)の手法はプログラムによりデータ転送を行うため、転送速度には限界があります。

(3)の DMAC は CPU からバス権を受け取り、データ転送を専門に行う周辺機能です。この DMAC によるデータ転送では、数Mバイト/S の転送も可能となります。プログラム上では、DMAC に起動をかけた後は、CPU はデータ転送に関し全く意識することなくプログラムを実行可能です。

CPU が行うデータ転送との大きな違いは、データ転送のため命令をフェッチする必要がない点です。たとえば、64180 の命令でデータ転送に適したブロック転送命令でも、1 バイトの転送に 14 ステートかかります。これに対し DMAC を使用すれば、6 ステートで1 バイトの転送を終了することができます。

64180 はこのように、高速データ転送に適した DMAC を 2 チャンネル内蔵しています。チャンネル 0 ではメモリ↔メモリ、メモリ↔I/O（メモリマップド I/O も含む）の転送が、チャンネル 1 ではメモリ↔I/O の転送が可能です（図 11・2）。

11・2 DMAC の 構 造

〔1〕 **DMAC のレジスタ** DMAC には、大別して、アドレスレジスタ、バイトカウントレジスタ、コントロール/ステータスレジスタの 3 つのレジスタが内蔵されています。64180 は 2 チャンネルの DMAC を内蔵しており、アドレスレジスタとバイトカウントレジスタは、チャンネルごとに独立に、またコントロールレジスタを共通にもっています (図 11・3)。

〔2〕 **アドレスレジスタ：チャンネル 0** チャンネル 0 ではメモリ↔メモリ、およびメモリ↔I/O のデータ転送が可能です。チャンネル 0 は DMA 転送を行う、データの転送元を示すソースアドレスレジスタ (20 ビット、I/O 空間の 20H, 21 H, 22 H 番地に配置)、およびデータの転送先を示す、ディスティネーションアドレスレジスタ (20 ビット、I/O 空間の 23 H, 24 H, 25 H 番地に配置) の 2 種類のアドレスレジスタをもっています。

〔3〕 **アドレスレジスタ：チャンネル 1** チャンネル 1 では、メモリ↔I/O のデータ転送が可能です。チャンネル 1 は、メモリの転送開始アドレスを設定するメモ

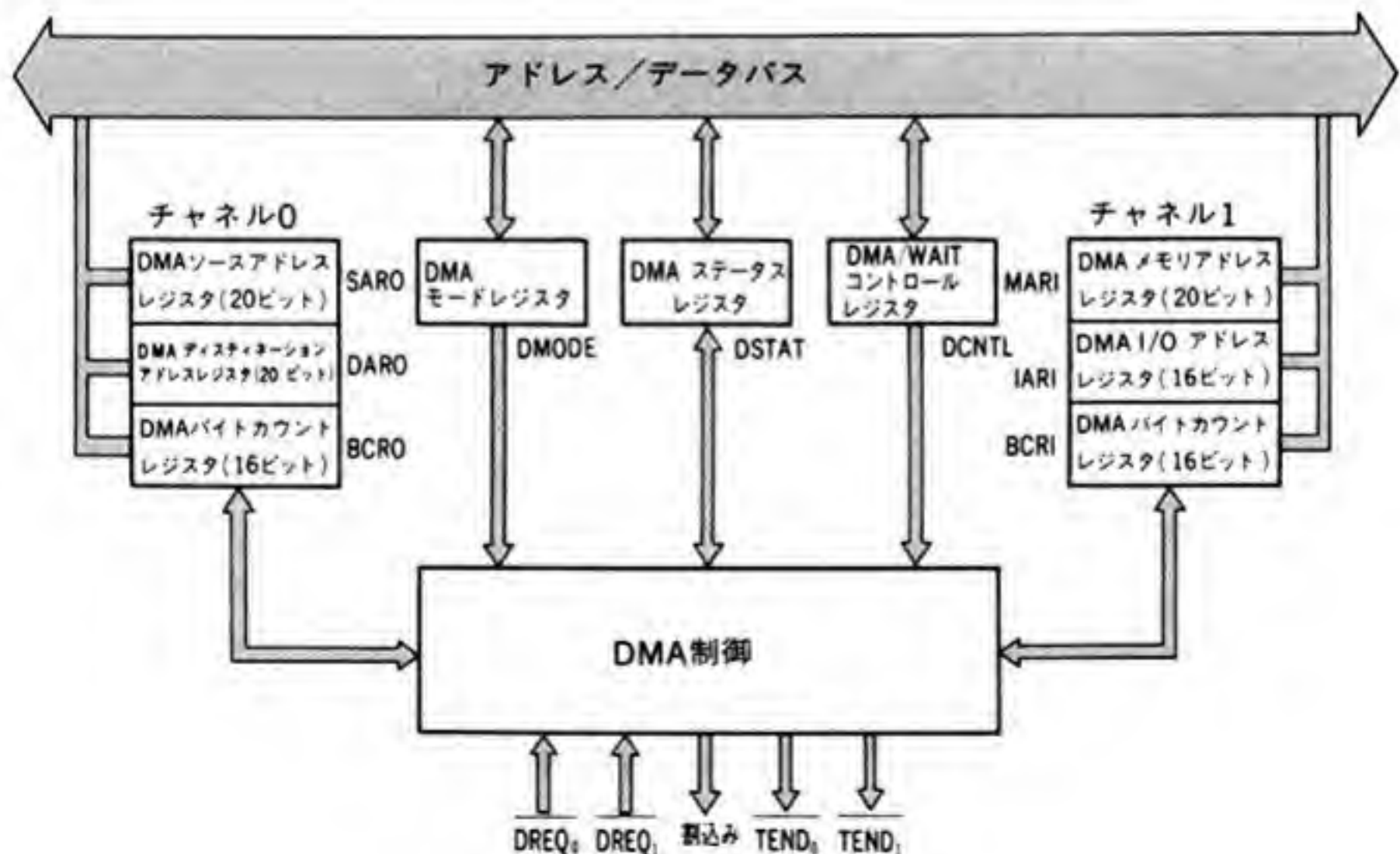


図 11・3 DMAC ブロック図

11. DMA コントローラ

リアドレスレジスタ（20ビット、I/O空間の28H、29H、2AH番地に配置）と、I/Oアドレスを設定するI/Oアドレスレジスタ（16ビット、I/O空間の2BH、2CHに配置）をもっています。

〔4〕 **バイトカウントレジスタ** DMA転送を行うデータの数（バイト数）を設定するレジスタです。たとえば、10バイトのデータ転送を行う場合、バイトカウントレジスタには0AHを設定します。また、バイトカウントレジスタに00Hを設定すると、最大の64 Kバイトの転送を行います。チャンネル0、1とも16ビットのバイトカウントレジスタをもっており、チャンネル0のバイトカウントレジスタはI/O空間の26H、27H番地に、チャンネル1のバイトカウントレジスタは2EH、2FHに配置されています。

〔5〕 **コントロール/ステータスレジスタ** チャンネル0、1の動作モードや割込みを制御するため、3バイトの制御レジスタをもっています。1つは、チャンネル0、1の動作のイネーブル、ディスエーブルや割込み制御を行うDMAステータスレジスタ（I/O空間の30Hに配置）、1つは、チャンネル0のメモリ↔メモリなどの動作モードを設定するためのDMAモードレジスタ（I/O空間の31Hに配置）、もう1つは、チャンネル1の動作モードを設定するためのDMA/WAITコントロールレジスタ（I/O空間の32Hに配置）の3つのレジスタです。

11・3 DMAC の転送モード

〔1〕 **メモリ↔メモリ** 64180 の内蔵した DMAC のチャンネル 0 では、メモリ↔メモリの DMA 転送が可能です。64180 の DMAC は、デュアルアドレス方式をとっています。デュアルアドレス方式は、従来 DMAC の主流であった、データのリードとライトを同時に行うシングルアドレス方式に対し、リードとライトが独立に発生する方式です。表 11・2 にシングルアドレス方式とデュアルアドレス方式の比較を示します。したがって、ウェイトステートが挿入されないとする、最小バスサイクルは 6 ステート（リード 3 ステート、ライト 3 ステート）です。最大転送レートは 1.3 M バイト/s ($f=8\text{ MHz}$) です。

表 11・2 シングルアドレス方式とデュアルアドレス方式

	バスサイクル	DACK 信号*	備 考
シングルアドレス方式	1 サイクル (1 マシンサイクル)	必 要	主にメモリ↔I/O の転送
デュアルアドレス方式	2 サイクル (2 マシンサイクル)	不 要	シングルアドレス方式に比べると速度は遅い

* DACK 信号…DMAC から出力されるチップセレクトに相当する信号

転送元、転送先のメモリアドレスの指定は、MMU を介さず 1 MB の物理アドレス空間のアクセスを行うため、20 ビットのアドレスレジスタにより行います。この機能により、論理空間上にない物理空間上のデータを転送することが可能となります。

メモリ↔メモリの転送には、2 つの転送モードがあります。1 つは、DMA 転送を開始すると、バイトカウンタレジスタが '0' になり、転送が終了するまで DMAC がバスを専有するバーストモードと、もう 1 つは、1 バイトの DMA 転送ごとに CPU の 1 マシンサイクルを実行し、これを交互に繰り返すサイクルスチールモードです。バーストモードでは、DMAC がバスを専有し CPU サイクルは実行されないため、転送が終了し CPU に制御が戻るまで割込みが受け付けられません。サイクルスチールモードでは CPU と DMAC が交互にバスを使用するため、DMA 転送を行いながら割込み処理を行うことも可能です（図 11・4）。

なお、バーストモードで DMAC がバスを専有している場合でも、リフレッシュコントローラによりリフレッシュサイクルを挿入可能で、DRAM の内容を保持

11. DMA コントローラ

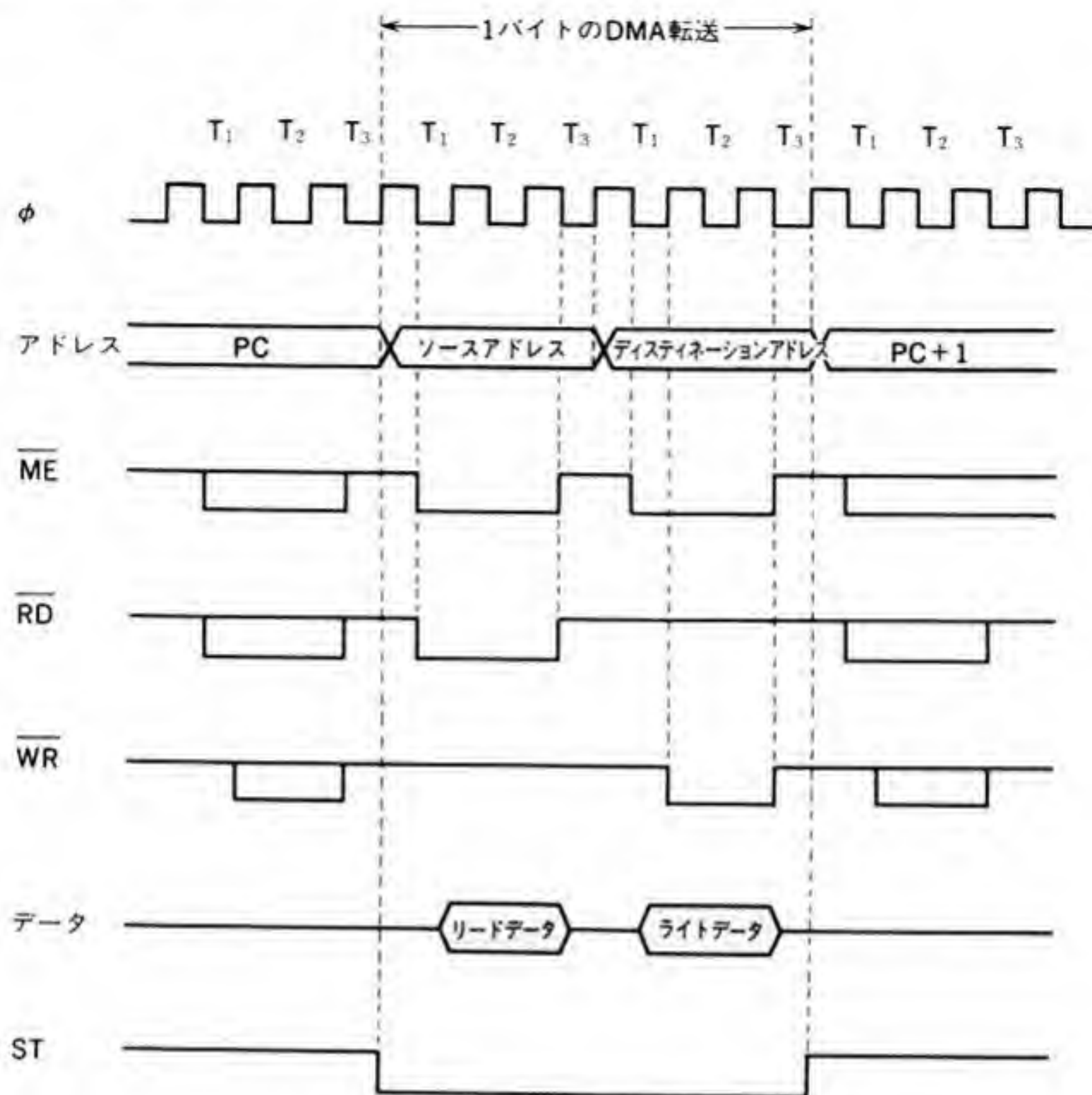


図 11・4 メモリ↔メモリの DMA 転送

することができます

〔2〕 **メモリ↔I/O** チャンネル 0、1 ともメモリ↔I/O の DMA 転送が可能です。また、チャンネル 0 ではメモリ↔メモリマップド I/O の転送も可能です。さらに、チャンネル 0 では内蔵の非同期シリアル (ASCI) との DMA 転送も可能です。メモリ↔I/O もデュアルアドレス方式をとっているため、メモリと I/O へのリードおよびライトサイクルが独立に発生します。I/O を DMA 転送の対象として選択する場合、CE 信号に代わり \overline{ACK} 信号 (DMA 転送の要求信号: \overline{DREQ} 信号に対する応答信号) を返す必要があります。デュアルアドレス方式の場合、メモリサイクルと独立に発生する I/O リード、あるいは I/O ライトサイクルに出力される I/O アドレスをデコードし、 \overline{ACK} 信号として使用します。

メモリ↔I/O の DMA 転送を行うためには、DMA の転送をイネーブルにした

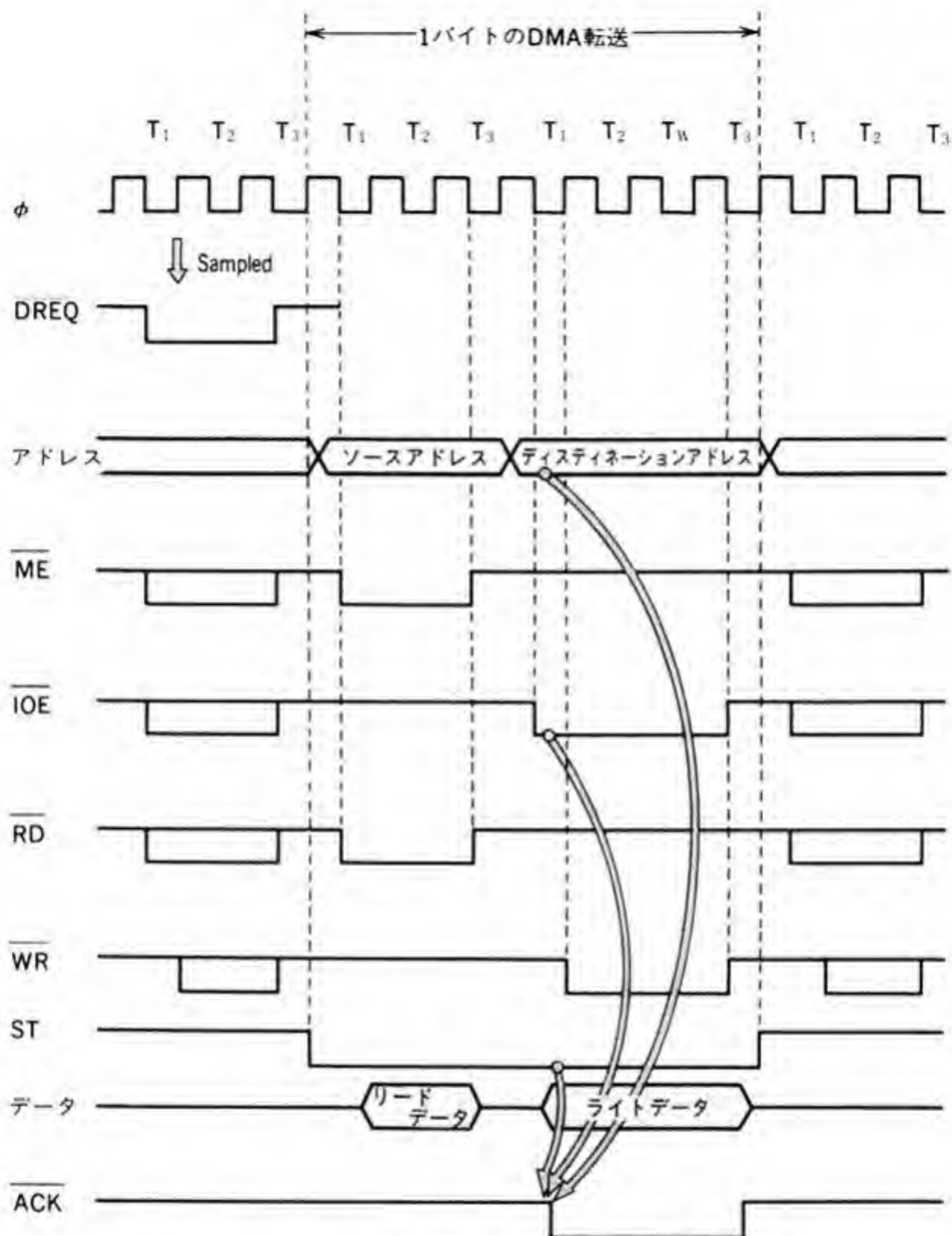


図 11・5 メモリ↔I/O のDMA転送（メモリ→I/Oの例）

状態（DEビットに‘1’をセットした状態）で、 $\overline{\text{DREQ}}$ 信号をアクティブ（Low）にします。この $\overline{\text{DREQ}}$ 信号は、コントロールレジスタの設定により、エッジセンスとレベルセンスを選択可能です。1バイトごとに同期をとって転送を行うにはエッジセンスが、連続して転送を行うにはレベルセンスが適しています（図 11・5）。

また、レベルセンスの場合要求はラッチされず、 $\overline{\text{DREQ}}$ 信号はサンプリング

11. DMA コントローラ

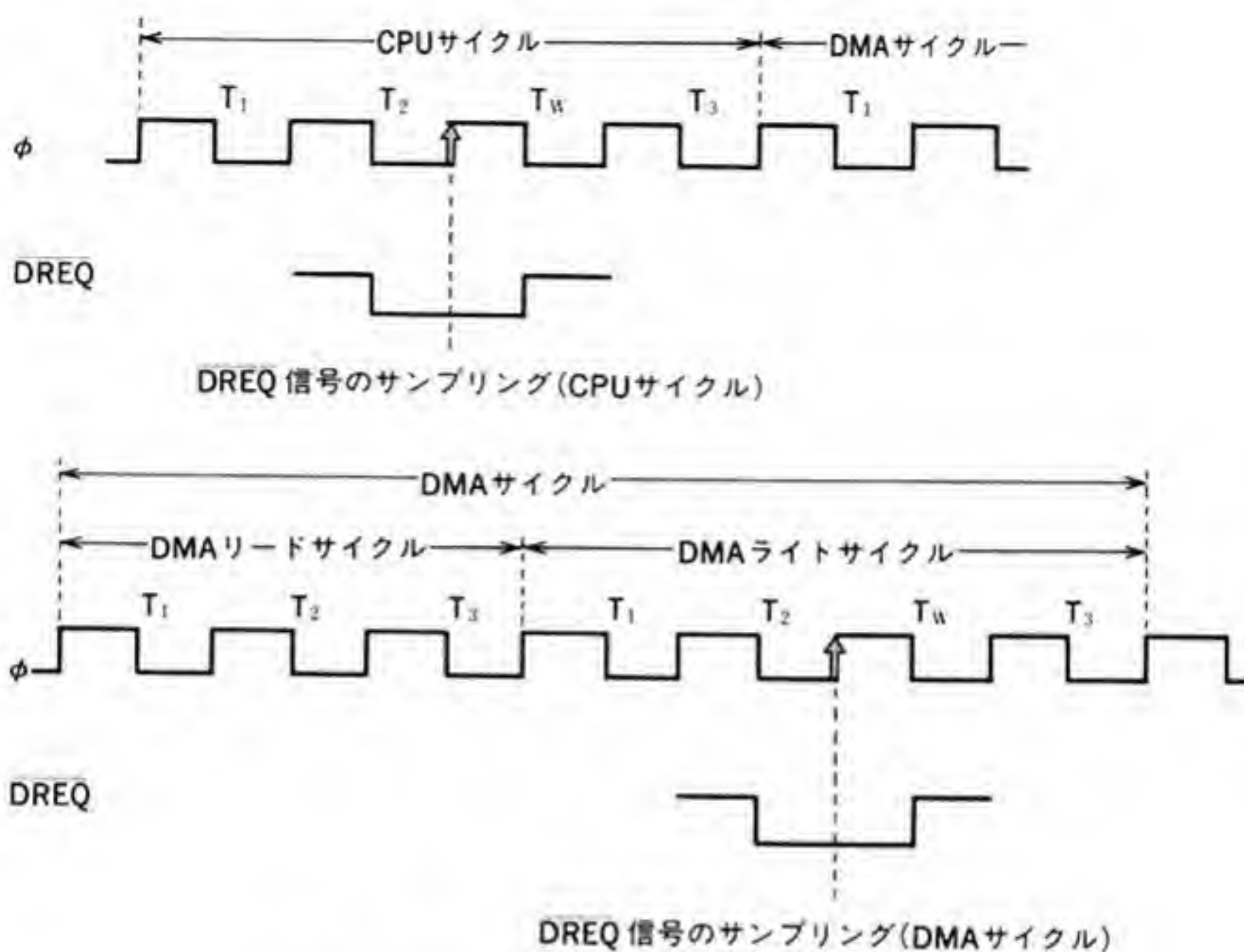


図 11・6 $\overline{\text{DREQ}}$ 信号のサンプリングタイミング

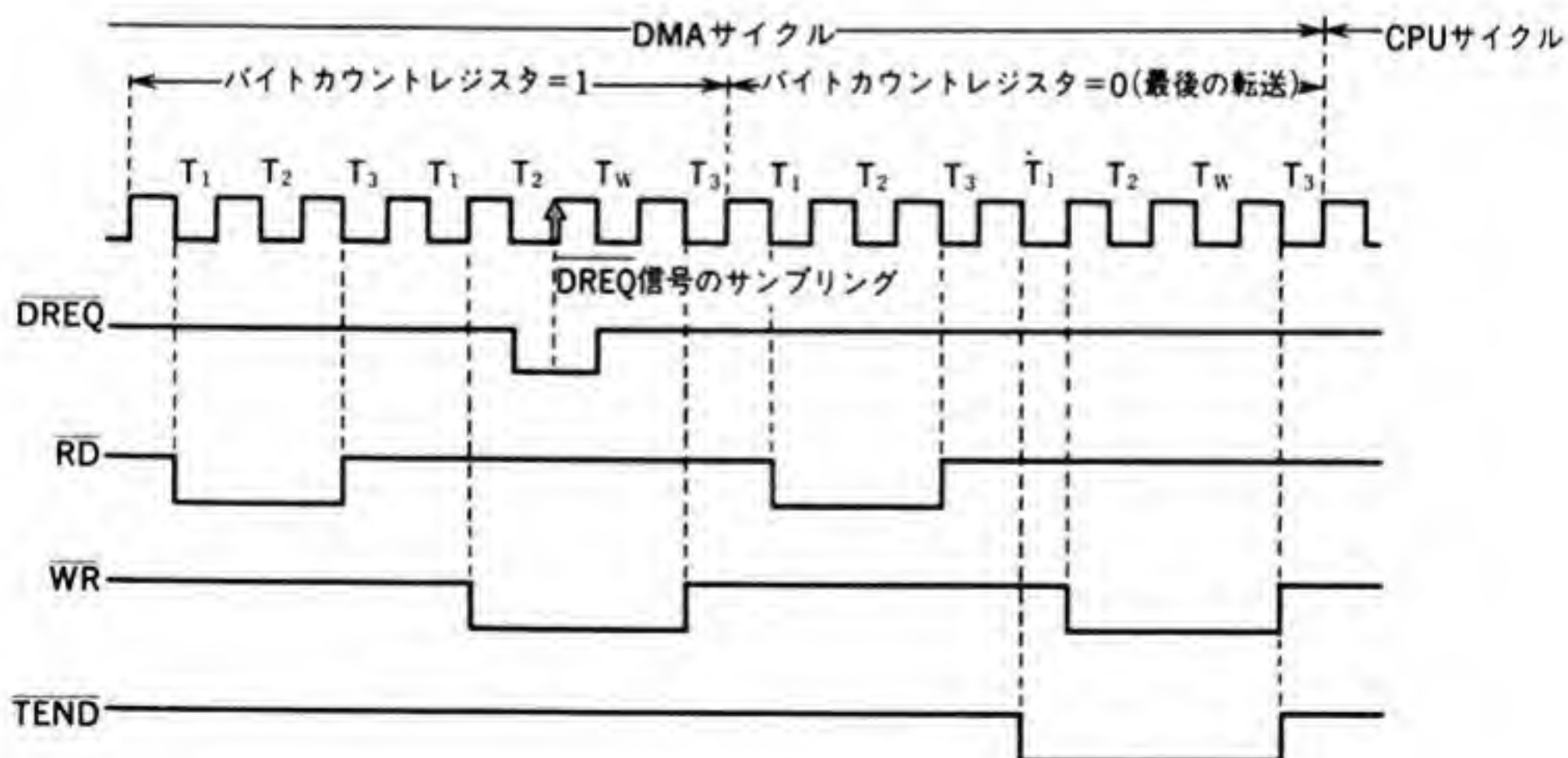


図 11・7 $\overline{\text{TEND}}$ 信号の出力

タイミングにアクティブになっている必要がありますが、エッジセンスの場合立下りエッジがラッチされ、サンプリングのタイミングまで保持されています。

$\overline{\text{DREQ}}$ 信号は、CPU の通常のマシンサイクル中は、マシンサイクルごとの T_3 サイクルの 1 つ前のクロックの立上りでサンプリングされ、DMA サイクル中は、DMA ライトサイクルの T_3 サイクルの 1 つ前のクロックの立上りでサンプリングされます。 $\overline{\text{DREQ}}$ 信号のサンプリングタイミングを図 11・6 に示します。

最大転送レートは、メモリサイクル 3 サイクル、I/O サイクル 4 サイクル (1 WAIT 挿入) の計 7 サイクルで 1 バイトのデータ転送が終了するため、1.1M バイト/s ($f=8\text{ MHz}$) を得ることができます。

転送が終了すると $\overline{\text{TEND}}$ 信号が出力され、I/O に対し DMA 転送が終了したことを知らせます。 $\overline{\text{TEND}}$ 信号は、最後の 1 バイトの転送のライトサイクル中アクティブ (Low 状態) になります。 $\overline{\text{TEND}}$ 信号の出力タイミングを図 11・7 に示します。

11・4 チャンネルの優先順位

DMA 転送の開始要求が、チャンネル 0、1 とともに同時に発生した場合、チャンネル 0 が優先的に受け付けられます。一方、どちらかのチャンネルが連続して DMA 転送を行っている場合、後から転送要求の発生したチャンネルは動作することができません。これには次の 2 つの場合が考えられます。

(1) チャンネル 0 がメモリ↔メモリで動作中の場合（バーストモード、サイクルスチールモードのいずれの場合も含む）…図 11・8 の場合

(2) チャンネル 0 または 1 が、メモリ↔I/O を連続して行っている場合…図 11・9 の場合

いずれの場合も動作中のチャンネルが優先され、後から転送要求が発生したチャンネルは動作することができません。

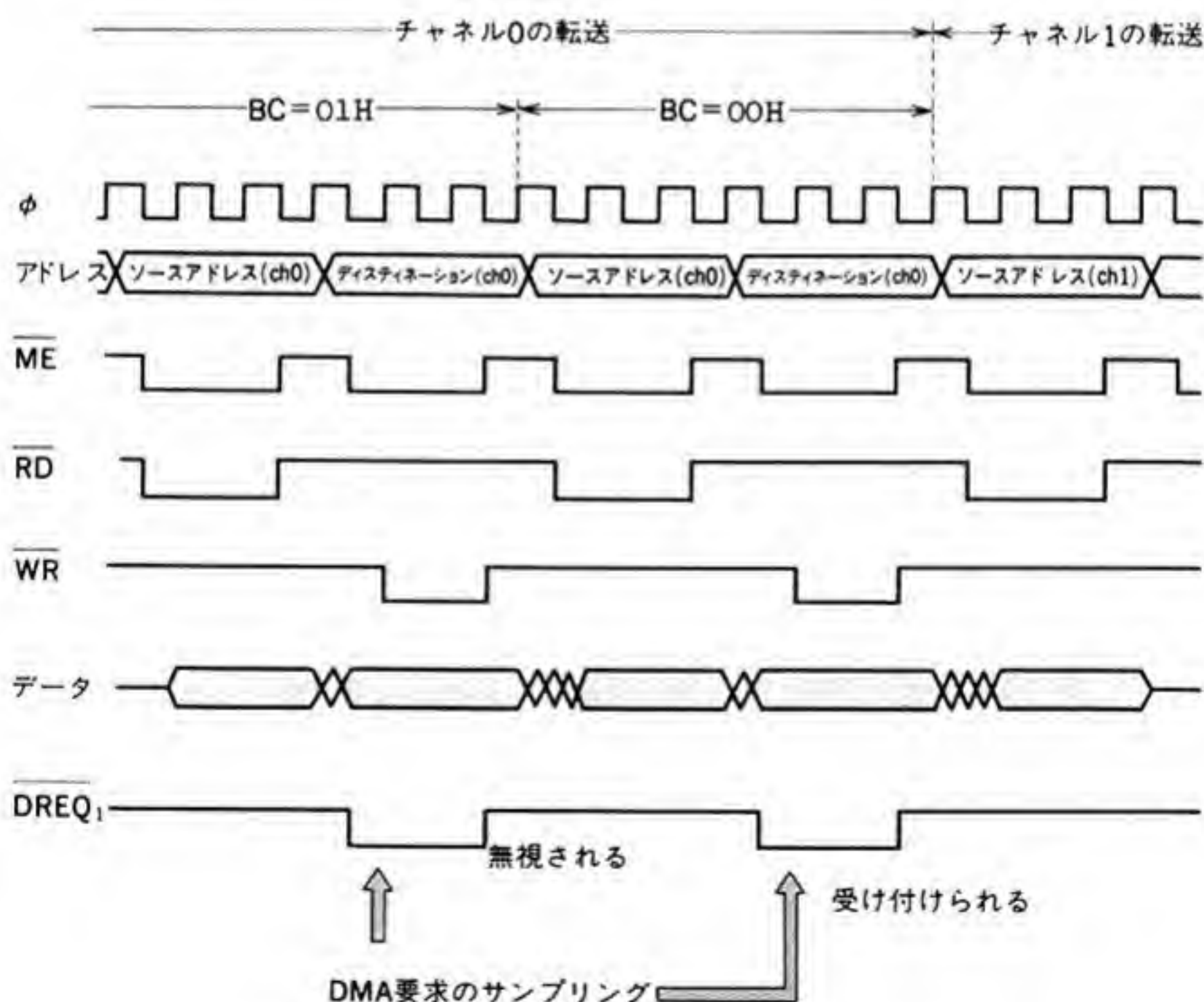


図 11・8 チャンネル 0 (メモリ↔メモリ) とチャンネル 1 の競合

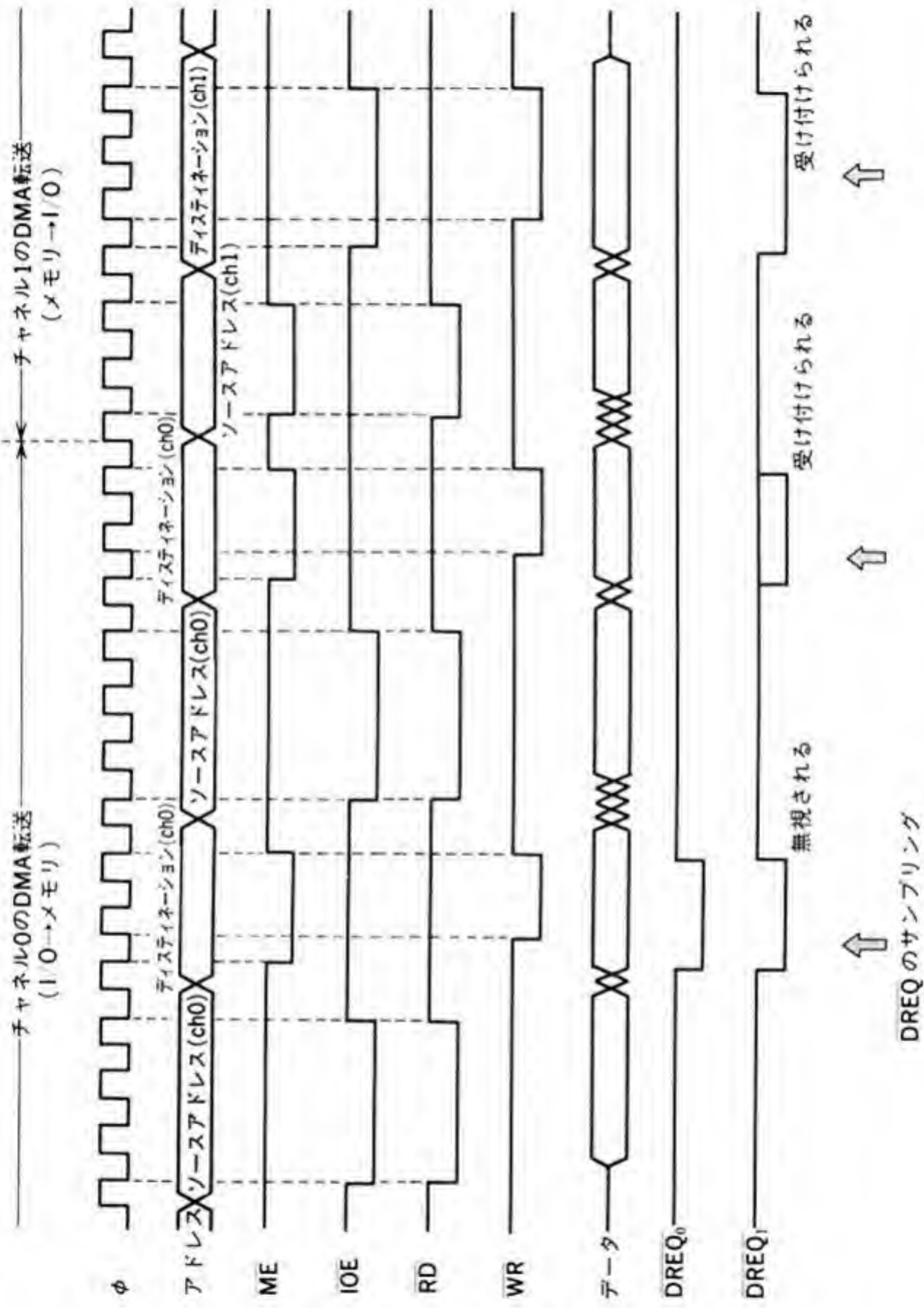


図 11・9 チャンネル 0 (メモリ→I/O) とチャンネル 1 の競合

11. DMA コントローラ

ただし、メモリ↔I/Oの転送の場合、動作中のチャネルの転送要求がとたえ、この間にもう一方のチャネルの転送要求が発生すれば、後から要求の発生したチャネルが動作可能となります。また、DMAの転送要求信号（ $\overline{\text{DREQ}}$ 信号）はエッジセンスとレベルセンスを選択可能ですが、エッジセンスを選択した場合、この要求は実際に転送が行われるまで1回だけ保持されます。したがって、動作中のチャネルの転送終了後、以前に入力され無視されていたエッジの $\overline{\text{DREQ}}$ 入力がサンプリングされ、DMA転送を行います。これに対し、レベルセンスでは要求を保持しませんから、動作中のチャネルの転送終了後、 $\overline{\text{DREQ}}$ 端子がアクティブ（Low状態）になっていないとDMA転送は行われません。

なお、メモリ↔メモリの転送の場合、転送が開始されると、サイクルスチールモードの場合もバーストモードの場合でも、チャネル0の転送が終了するまでチャネル1は動作することができません。

11・5 DMAC とブロック 転送命令

64180 の DMA 転送では、メモリ↔メモリのデータ転送が可能です。一方、64180 はデータ転送を行うため、ブロック転送命令をもっています。ここでは、DMA 転送とブロック転送命令の転送速度の比較をしてみましょう。

DMAC のアドレスレジスタの設定には、メモリアドレスと I/O アドレスが 1 バイト設定ごとにプラス 1 される、OTIMR 命令を使用します。ただし、設定値はメモリ空間上のテーブルに連続して配置されているとします。次に、OUTO 命令を用いて、DMA ステータスレジスタ、DMA モードレジスタ、DMA/WAIT コントロールレジスタの設定を行います。このプログラムを実行するには、201 ステート必要です。したがって、物理空間上でのメモリ↔メモリの n バイトの転送を行うと、 $6n+201$ ステートとなります。

次に、ブロック転送命令を使用した場合を考えます。HL レジスタに転送元のメモリアドレス、DE レジスタに転送先のメモリアドレス、BC レジスタに転

LD HL, CNST0	9	←	アドレスレジスタに設定する設定値が格納さ
LD BC, CNST1	9		れているメモリアドレスの先頭番地
OTIMR	$\begin{cases} 16 & \text{不成立} \\ 14 & \text{成立} \end{cases}$	$16 \times 7 + 14$	B: 設定するレジスタのバイト数(8 バイト) C: DMAC のレジスタの先頭アドレス(20H 番地)
LD A, CNST2	6		
OUTO(DCNTL), A	13		
LD A, CNST3	6		
OUTO(DMODE), A	13		
LD A, CNST4	6		
OUTO(DSTAT), A	13		DMA イネーブル

201 ステート

(1) DMA 転送

LD HL, CNST5	9	転送元のメモリアドレス
LD DE, CNST6	9	転送先のメモリアドレス
LD BC, CNST7	9	転送バイト数
LDIR	$\begin{cases} 14 & \text{不成立} \\ 12 & \text{成立} \end{cases}$	$14(n-1)+12$

14n+25 ステート

(2) ブロック転送

図 11・10 DMAC とブロック転送命令

11. DMA コントローラ

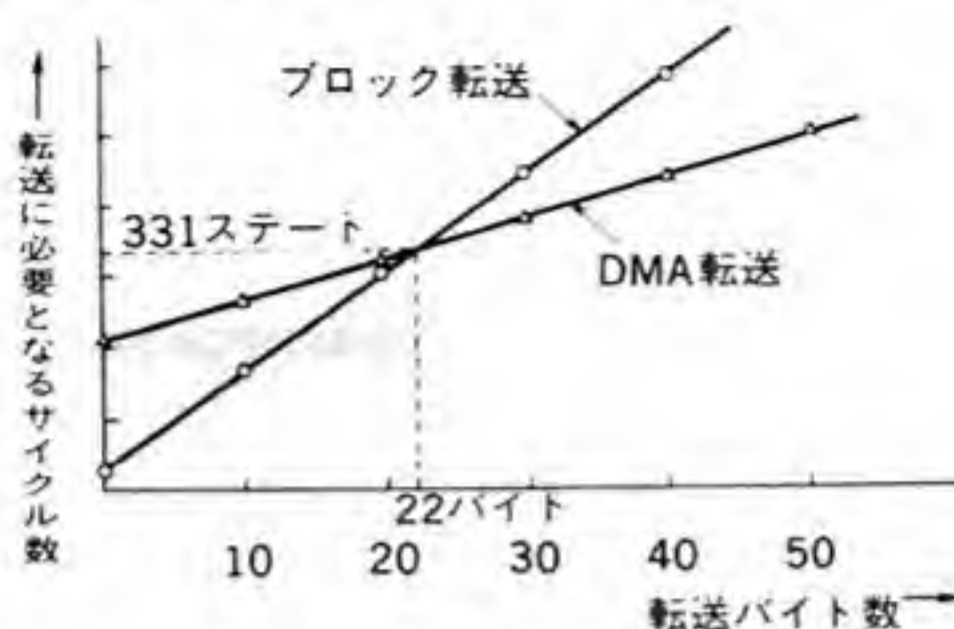


図 11・11 DMA 転送とブロック転送のサイクル数

表 11・3 DMAC とブロック転送命令の比較

	DMAC	ブロック転送命令
転送の対象となる空間	物理空間 (1 M バイト)	論理空間 (64 K バイト)
連続転送可能なバイト数	64 K バイト	←
1 バイトの転送に要するステート数	6 ステート	14 ステート
n バイトの転送に必要なステート数	$6n + 201$ ステート	$14n + 25$ ステート

送バイト数を設定し、ブロック転送命令 (LDIR 命令など) を実行します。n バイトの転送を実行するには、 $14n + 25$ ステート必要となります。実際のプログラムを図 11・10 に示します。

したがって、図 11・11 に示すとおり 22 バイト以上の転送では、DMAC を使用したほうが速く転送が行えます。

表 11・3 に DMAC とブロック転送命令の比較を示します。DMA 転送とブロック転送命令の大きな違いは、前者は物理空間上のデータ転送を対象としているのに対し、後者は論理空間上でのデータ転送を対象としている点です。たとえば、論理空間上で連続しているコモンエリア 1 とバンクエリアが、物理空間上では不連続な場合もあります。このような場合は、DMAC よりもブロック転送命令を使用するほうが得策です。これに対し、論理空間上にない物理空間上のデータを参照する場合は、DMAC により直接データを論理空間上にストアするのが便利です。

11・6 DMAC の使用例

〔1〕 メモリ↔メモリ DMAC のチャンネル 0 でメモリ↔メモリの転送を行うためには、次のような手続きを行います。

- (1) 転送元の転送開始アドレスを SAR0 (I/O 空間の 20, 21, 22H) に設定
 - (2) 転送先の転送開始アドレスを DAR0 (I/O 空間の 23, 24, 25H) に設定
 - (3) 転送するデータのバイト数を BCR0 (I/O 空間の 26, 27H) に設定
 - (4) メモリ↔メモリの指定、アドレスの増減、およびサイクルスチール、バーストモードの転送モードを DMA モードレジスタ (I/O 空間の 31H) に設定
 - (5) DMA ステータスレジスタ (I/O 空間の 30H) に設定を行い、DMA 転送をイネーブルにする。また DMA 転送終了後、割り込みを発生することも可能
- DMA 転送をイネーブルにした I/O ライトサイクル終了後、次の命令の 1 マシンサイクルを実行し、DMA 転送が開始されます。DMAC はレジスタが多く(ただ



図 11・12 メモリ↔メモリ転送のプログラム例 (チャンネル 0)

11. DMA コントローラ

し連続に配置されている), 64180 の新規命令の OTIMR 命令や OUTO 命令を使用すると便利です。図 11・12 にフローチャートと実際のプログラムを示します。

〔2〕メモリ↔I/O チャンネル 0, 1 ともメモリ↔I/O の転送が可能です。ここでは、チャンネル 1 でメモリ→I/O の転送を行う手順を示します。

- (1) 転送元のメモリアドレスを MAR1 (I/O 空間の 28, 29, 2AH) に設定
- (2) 転送先の I/O アドレスを IAR1 (I/O 空間の 2B, 2CH) に設定
- (3) 転送するデータのバイト数を BCR1 (I/O 空間の 2D, 2EH) に設定
- (4) DREQ 信号のセンス (エッジセンス, レベルセンス) の指定, ソース, ディスティネーションの指定, およびメモリアドレスの増減の指定を, DMA/WAIT コントロールレジスタ (I/O 空間の 32H) に設定
- (5) DMA ステータスレジスタに設定を行い, DMA をイネーブルにする。また割込みも使用可能

この状態で DREQ 信号がアクティブ (Low 状態) になると, メモリ→I/O の DMA 転送が行われます。図 11・13 にフローチャートとプログラム例を示します。



図 11・13 メモリ↔I/O 転送のプログラム例 (チャンネル 1)

第Ⅲ編

応 用

第Ⅲ編では、64180 を使って実際にシステムを組む場合のインタフェース例を述べます。64180 は周辺 LSI の機能を取り込んでいますが、マルチチップマイコンのため、メモリ、I/O を接続してシステムを組みます。そこで、12 章では SRAM、DRAM などのメモリとのインタフェース方法、13 章では各種周辺 LSI とのインタフェース方法を述べます。14 章では、64180 の新規命令の使用例を述べます。15 章では 64180 を使用した具体的なシステムとして、180 カードパソコンについて述べます。180 カードパソコンは葉書サイズのボードに、64180、256 KDRAM、EPROM、FDC を搭載した CP/M PLUS システムです。

12. メモリ インタフェース

64180 は MMU を通して、1 M バイトもの大容量メモリを管理することができます。この章では、64180 とメモリ (SRAM, DRAM, EPROM) との具体的なインタフェース例を述べます。64180 はリフレッシュコントローラを内蔵しているため、DRAM と也容易に接続できます。また、最近注目されてきている DRAM のように大容量メモリでありながら、SRAM のように使いやすい擬似 SRAM とのインタフェース例も述べます。

12・1 SRAM の 接 続

64180 は SRAM と簡単に接続できます。HM62256 (32 K ワード×8 bit) との接続の場合、 \overline{CS} 信号はアドレスと \overline{ME} 信号により生成します。 \overline{OE} 信号は \overline{RD} 信号に、 \overline{WE} 信号は \overline{WR} 信号に接続します。図 12・1 では、SRAM は 64180 の物理アドレス 08000H 番地～0FFFFH 番地に配置します。

メモリアクセス時のクリティカルパスは、オペコードフェッチ時です。オペコードフェッチ時は T_3 サイクルの立上りでデータが取り込まれます。 $\phi=6.144$ MHz 時、ウェイトサイクルなしでアクセスタイムの許容範囲は 144 ns (\overline{CS} 信号デコード回路の遅延を含む) 以下で、アクセスタイム 120 ns の SRAM が使えます。 $\phi=8$ MHz 時では、ウェイトサイクルなしでアクセスタイムの許容範囲は 107 ns (\overline{CS} 信号デコード回路の遅延を含む) 以下で、アクセスタイム 85 ns の SRAM が使えます。

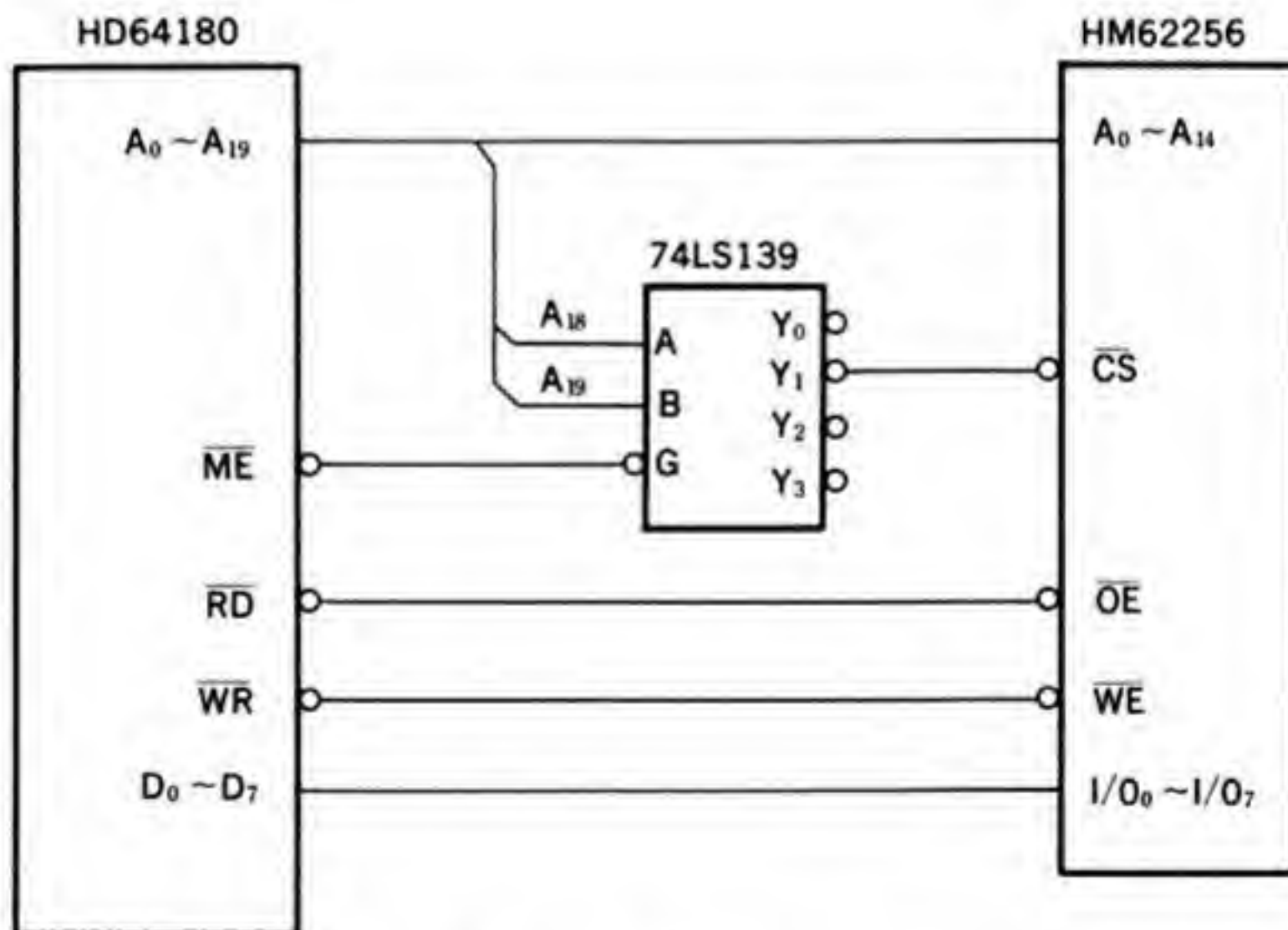


図 12・1 SRAM の接続

74LS139

G	B	A	Y ₀	Y ₁	Y ₂	Y ₃
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

12・2 擬似 SRAM の接続

擬似 SRAM は、SRAM と同様に使えること、かつ DRAM のように安価であることからポピュラーになりつつあります。64180 に擬似 SRAM を接続した例を図 12・2 に示します。擬似 SRAM は、DRAM と同様にリフレッシュを行わなければなりませんが、64180 はリフレッシュ機能を内蔵しているため、回路が簡単になります。HM658128 は 9 bit のリフレッシュアドレスが必要ですが、64180 は 8 bit リフレッシュアドレスしか出力しないため、本来なら 1 bit 分追加する外部回路が必要です。その対策として、擬似 SRAM がもっているリフレッシュアドレスを必要としないオートリフレッシュモードを使用します。このモードでは HM658128 内部のリフレッシュカウンタにより、 \overline{CE} ='High' 時に \overline{RFSH} ='Low' にするだけで、内部でリフレッシュが実行されます。したがって 64180 を使う場合、8 ms 内に 512 回リフレッシュサイクルが入るようにリフレッシュサイクルの間隔を設定するだけで、擬似 SRAM のリフレッシュが行えます。たとえば $\phi=6.144$ MHz の場合、リフレッシュ間隔を 80 ステートに設定します。

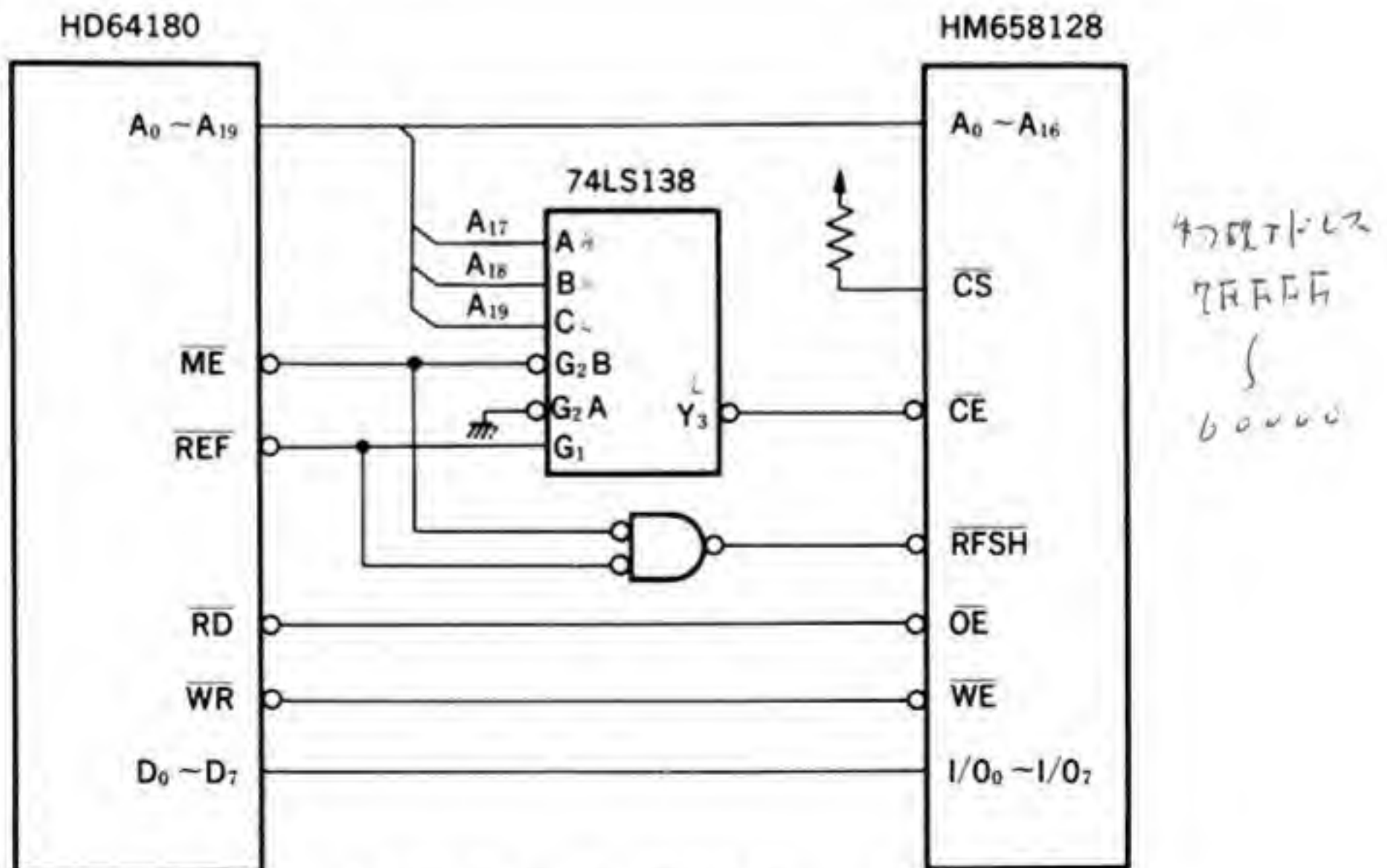


図 12・2 擬似 SRAM の接続

12・3 DRAM の 接 続

DRAMを接続するためには、アドレスのマルチプレクサ、 $\overline{\text{RAS}}$ 、 $\overline{\text{CAS}}$ 信号の生成、リフレッシュサイクルの発生が必要です。64180 はリフレッシュコントローラを内蔵しているため、通常のマイコンシステムで必要なリフレッシュコントロール用回路（リフレッシュアドレス発生用カウンタ、CPU へのバス権要求回路など）が不要になります。64180 に 256KDRAM を接続した例を図 12・3 に示します。

DRAM のアドレスは、ロウアドレスとコラムアドレスを時分割で入力します。そのため、アドレスマルチプレクサを使います。 $\overline{\text{RAS}}$ 信号の立下りでロウアドレスを確定しますが、64180 の $\overline{\text{ME}}$ 信号をそのまま $\overline{\text{RAS}}$ 信号として使用できます。この場合、 $\overline{\text{ME}}$ 信号の 'High' 時間およびアドレスセットアップ時間に注意します。図 12・4 に DRAM アクセスタイミングを示します。アドレスマルチプレクサ切替え信号と $\overline{\text{CAS}}$ 信号は、通常ディレイ回路で複雑になるのですが、64180

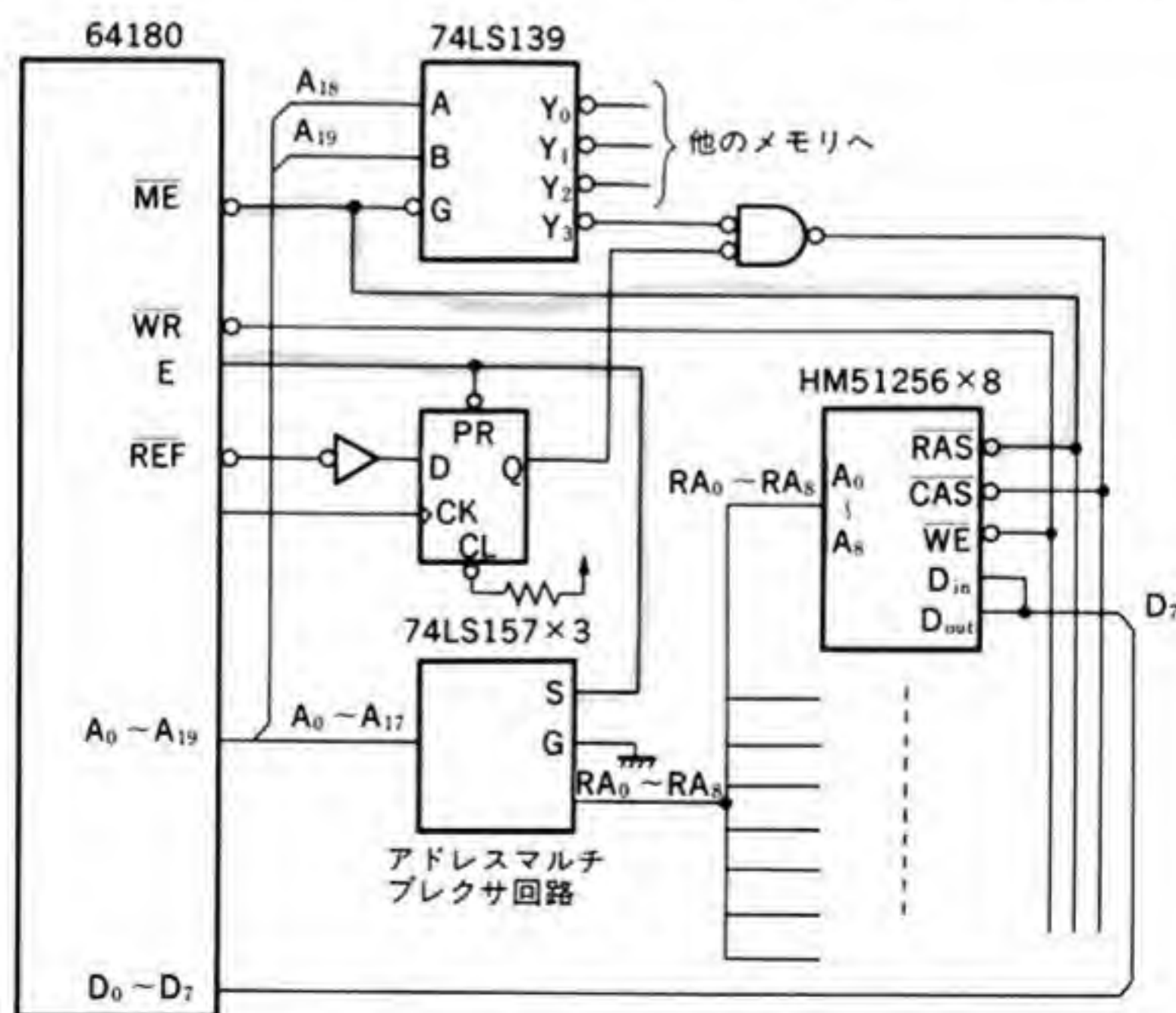


図 12・3 DRAM の接続例

から出力されているE信号を使用すると簡単な回路で生成できます。本来E信号は、68系周辺LSIを接続するための信号ですが、DRAMコントロールに流用しています。ME信号によるRAS信号の立下りでロウアドレスを確定し、E信号の立上りでカラムアドレスに切り替え、次の ϕ 信号の立上りでCAS信号を立ち下げ、カラムアドレスを確定します。この例ではCAS信号が遅れるため、1ウェイトサイクル挿入します。

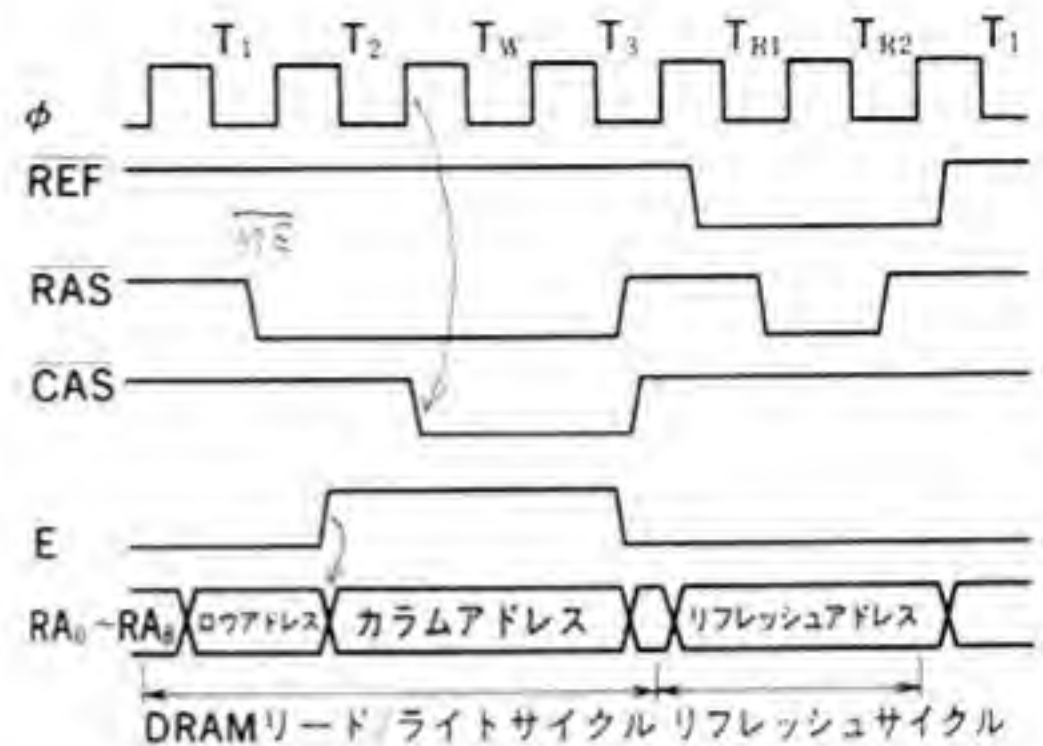


図 12・4 DRAM アクセスタイミング

DRAMのリフレッシュは、RASオンリリフレッシュモードで行います。この場合リフレッシュアドレスを入力しますが、64180内蔵のリフレッシュコントローラが、8bitのリフレッシュアドレスを発生します。リフレッシュサイクルの間隔は、動作周波数によりリフレッシュコントロールレジスタを設定して決めます。たとえば、 $\phi=6.144\text{ MHz}$ の場合、80ステート間隔に設定します。64180内蔵のリフレッシュコントローラは8bitのリフレッシュアドレスを出力するため、256リフレッシュサイクル/4msの256KDRAMしか接続できません。512リフレッシュサイクル/8msの1MDRAMを接続するためには、図12・5に示すようなリフレッシュアドレスを1bit追加する回路を外付します。

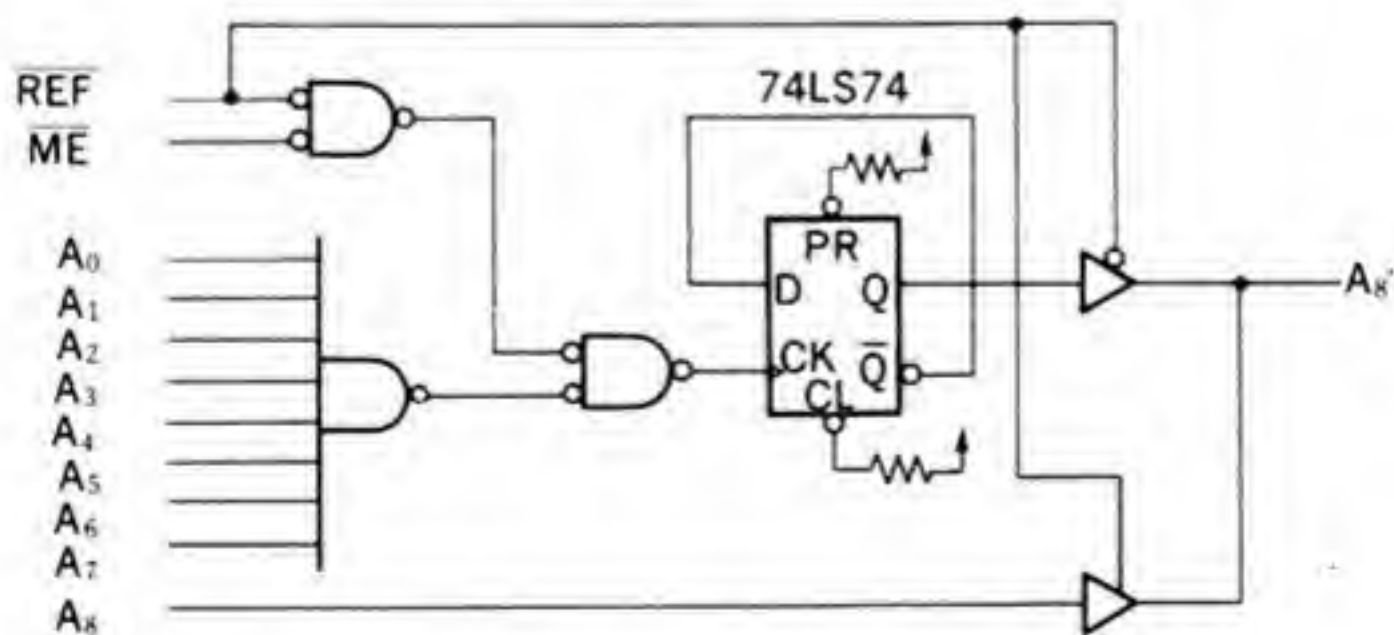


図 12・5 1M DRAM 接続時のリフレッシュアドレス追加回路

12・4 EPROM の接続

EPROMの \overline{CE} 信号はアドレスと \overline{ME} 信号で生成します。EPROMはリードオンリのため、誤ってEPROMにデータをライトしようとしても、データが衝突しないように64180の \overline{RD} 信号をEPROMの \overline{OE} 信号に接続してください。SRAMの場合と同様に、オペコードフェッチ時のアクセスタイムの許容範囲は $\phi=6.144\text{ MHz}$ で 144 ns 以下、 $\phi=8\text{ MHz}$ で 107 ns 以下（いずれもウェイトサイクルなしでデコード回路の遅延を含む）になります。最近では、アクセスタイムが 100 ns 以下の高速EPROMが出始めていますが、大部分のEPROMのアクセスタイムは 200 ns 以上です。そのため、ウェイトサイクルを挿入しなければなりません。64180内蔵のウェイトコントロール機能を使うと、外付ハードウェアなしでEPROMをアクセスできます。

図12・6では、消費電流を抑えるため、アクセスタイミングを \overline{CE} 、 \overline{OE} 両信号で制御しています。EPROMはリードオンリのため、 $\overline{CE}=\text{'Low'}$ 固定とし、 \overline{OE} 信号のみで制御することができます。この場合、アクセスタイムを短縮でき、ウェイトサイクルを挿入しなくてもよくなります。ただし、 $\overline{CE}=\text{'Low'}$ のままだと消費電流は増えたままです。特にCMOSタイプのEPROMでは、顕著な差が現れます。

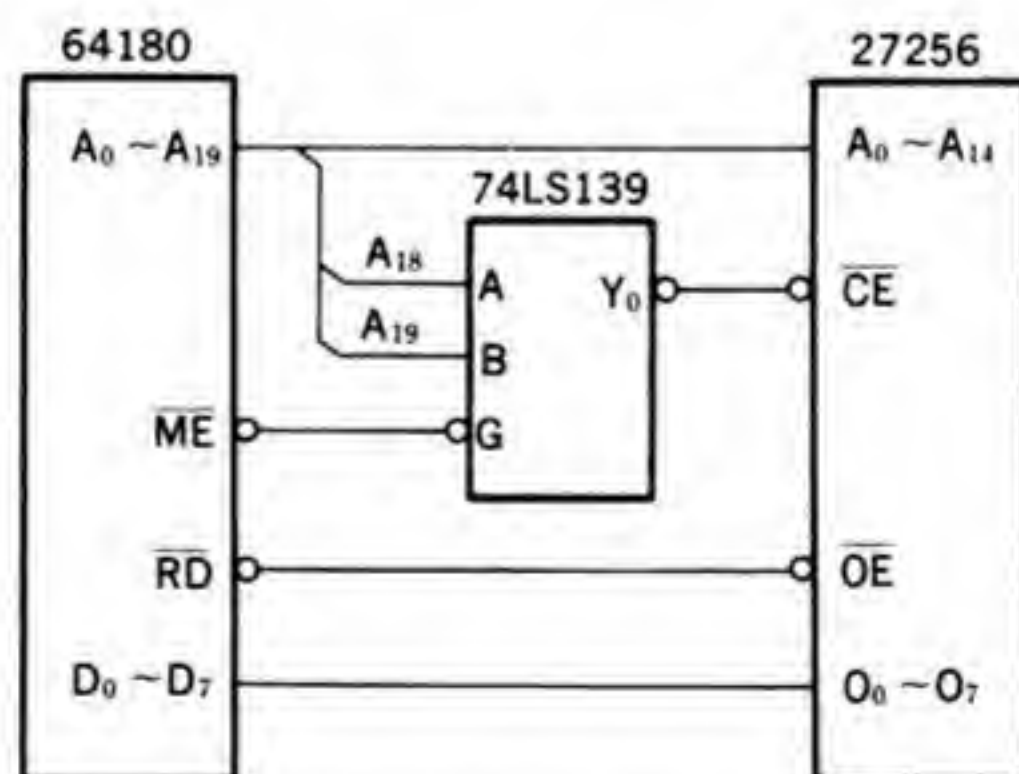


図 12・6 EPROM の接続

12・5 $\overline{\text{WAIT}}$ 信号発生回路

64180 内蔵のウェイトコントロール機能は、外付ハードウェアなしで中低速メモリをアクセスできるので便利です。ただし、メモリアクセス時に必ず設定した数だけウェイトサイクルが入るため、EPROMとSRAMが混在した場合など、EPROMをアクセスしたときはウェイトサイクルが入り、SRAMをアクセスしたときはウェイトサイクルが入らないようにすることは、ソフトウェアだけでは不可能です。64180の $\overline{\text{WAIT}}$ 信号にLowを入力すると、その期間ウェイトサイクルを挿入できます。図12・7のような、EPROMの $\overline{\text{CE}}$ 信号が'Low'になったときだけ $\overline{\text{WAIT}}$ 信号を'Low'にする外付回路を付加すると、選択的なウェイトサイクル挿入ができます。図12・8にウェイトステートタイミングを示します。

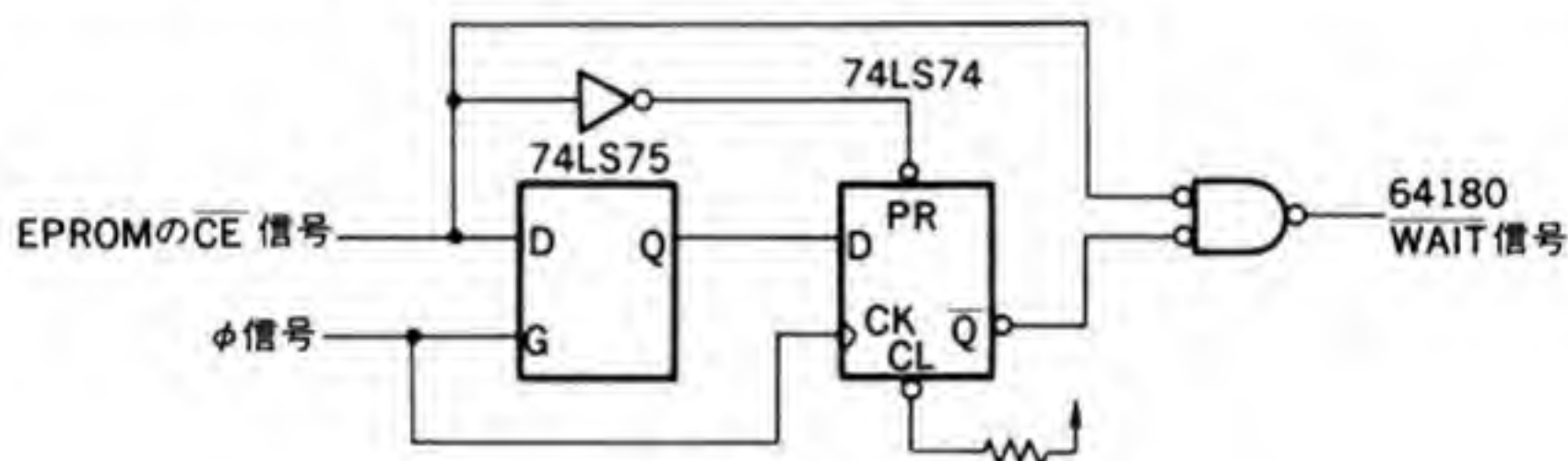
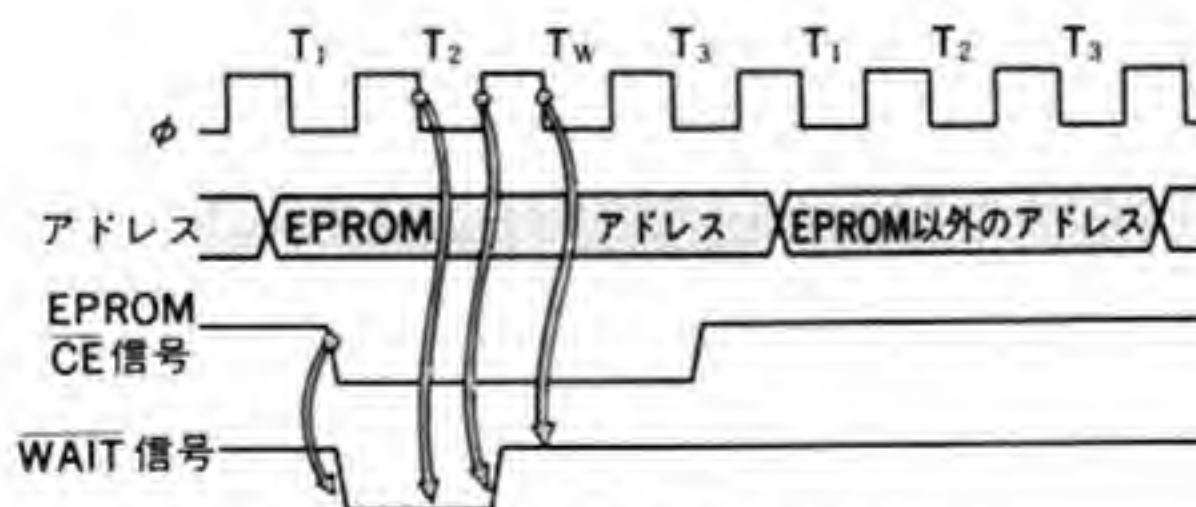
図 12・7 $\overline{\text{WAIT}}$ 信号発生回路

図 12・8 ウェイトステートタイミング

13. I/O インタ フェース

64180 は、DMAC, ASCI, タイマなどの周辺機能を内蔵していますが、その他に各種 I/O を外付できます。8255, 8251 などの 80 系周辺 LSI, Z80-PIO, SIO などの Z80 系周辺 LSI, 6845, 6321 などの 68/63 系周辺 LSI とのインタフェース例を述べます。Z80 系周辺 LSI を接続する場合、64180Z バージョンを使用します。68/63 系周辺 LSI 用には、64180 から出力されている E 信号を使用します。

13・1 80 系周辺 LSI との接続

80 系周辺 LSI は、 \overline{CS} 、 \overline{RD} 、 \overline{WR} 信号で制御します。図 13・1 では、80 系タイマ 8253 を例にとり説明します。 \overline{RD} 、 \overline{WR} 信号に対する \overline{CS} 信号のセットアップ時間を確保するため、 \overline{CS} 信号はアドレスのみから生成します。80 系周辺 LSI 用の \overline{RD} 、 \overline{WR} 信号は、64180 の \overline{IOE} 信号と \overline{RD} 、 \overline{WR} 信号の AND をとって生成します。

80 系周辺 LSI では \overline{CS} 、 \overline{RD} 、 \overline{WR} 信号の幅でアクセスが規定されています。そのため、64180 の動作周波数が高くて、80 系周辺 LSI をアクセスするときだけウェイトサイクルを挿入することにより 64180 を接続できます。64180 は内蔵ウェイトコントローラにより、ソフトウェアで I/O 空間アクセス時に 1～4 ウェイトサイクル挿入できます。この機能を使用すると、64180 が高速動作時も外部ウェイト回路なしで 80 系周辺 LSI を接続できます。 $\phi=6.144\text{ MHz}$ で 8253 が 5 MHz バージョンの場合、1 ウェイトサイクル挿入します。図 13・2 にタイミ

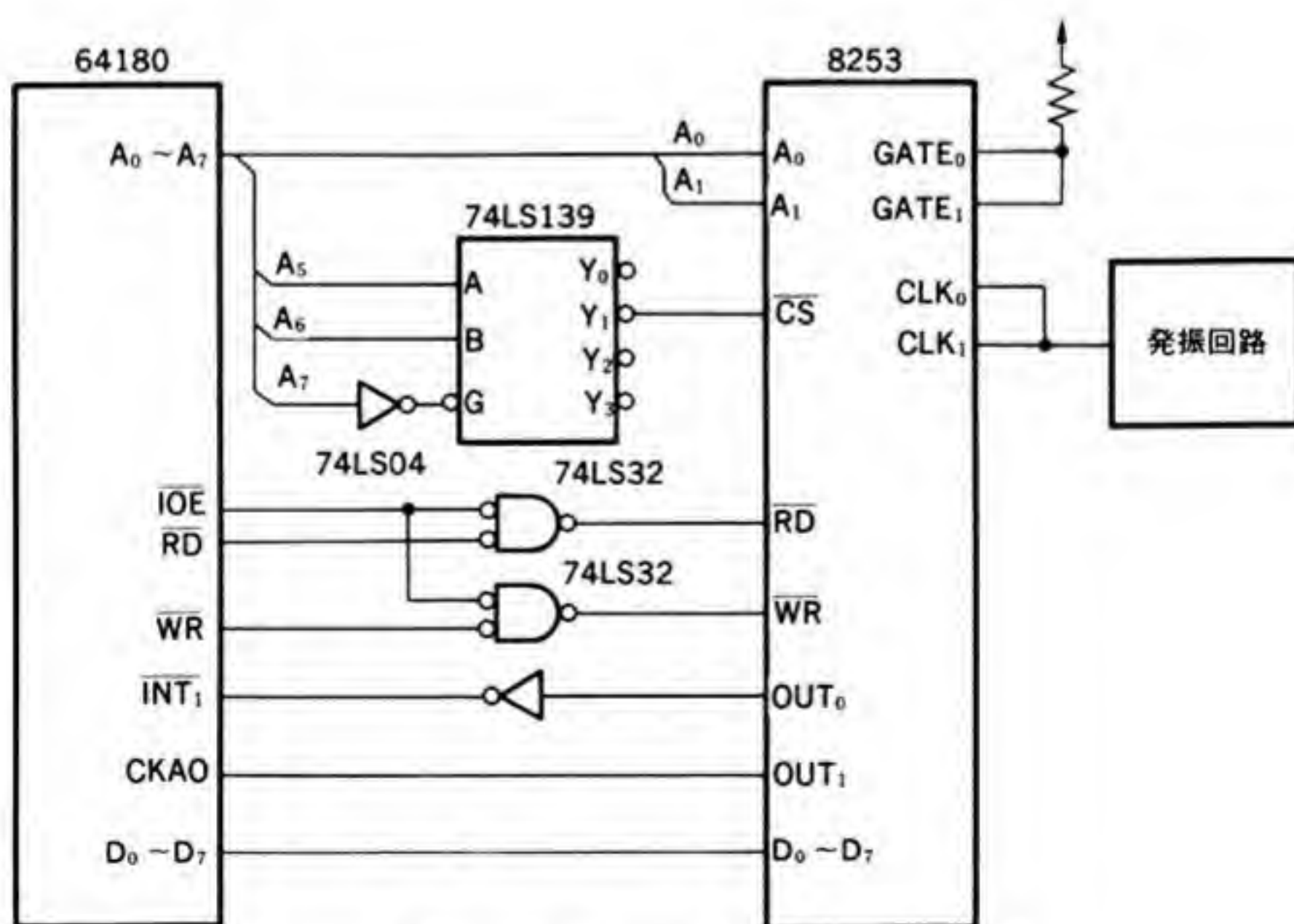


図 13・1 80 系周辺 LSI の接続例

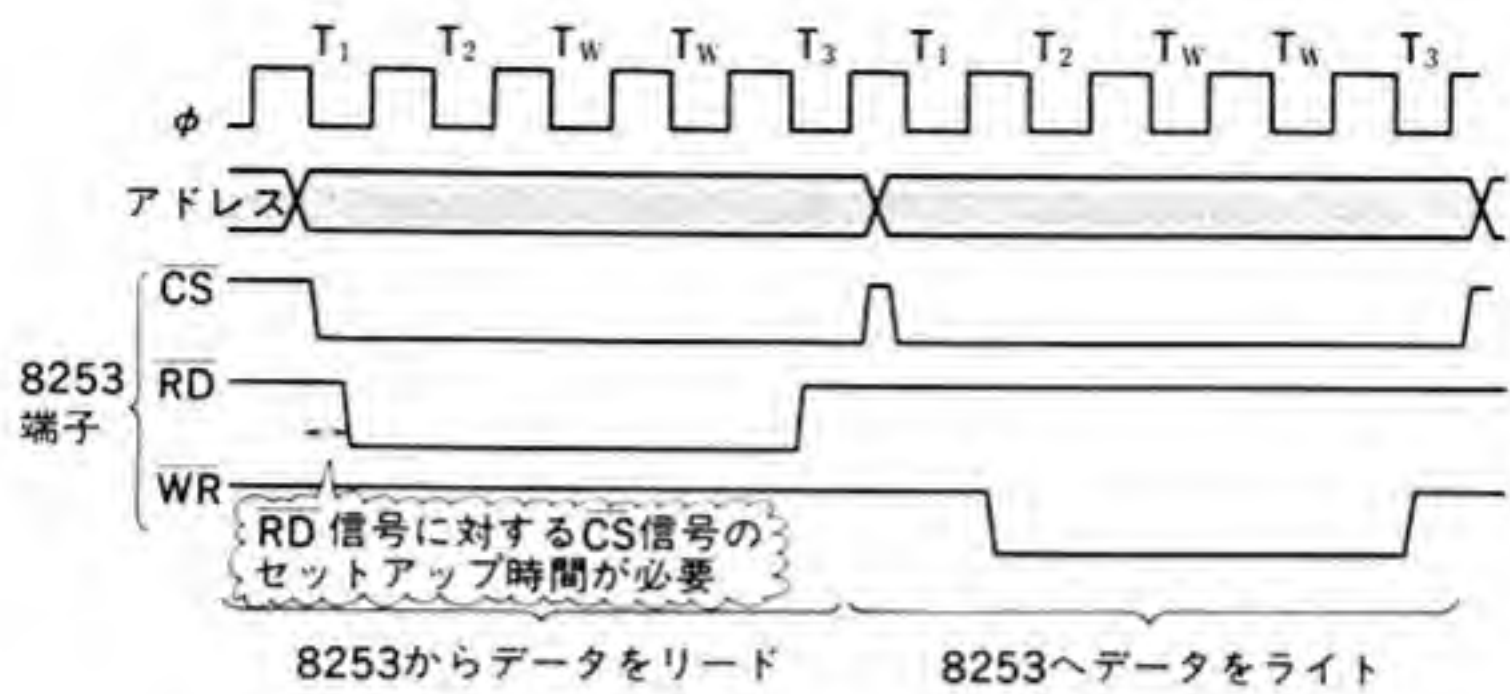


図 13・2 8253のアクセスタイミング

ング図を示します。なお、この場合、各信号のセットアップ、ホールド時間に注意しなければなりません。

周辺 LSI の種類

8ビットマイコンの周辺LSIは、80系、Z80系、68/63系に分けられます。表13・1に周辺LSIの種類を示します。通常、バスタイミングの違いから、80系・Z80系と68系の周辺LSIを混同して使用することは困難でした。しかし、64180は68系周辺LSI用にE信号を出力しているため、68系周辺LSIを容易に接続できます。

表 13・1 周辺LSIの種類

	80系	Z80系	68/63系
パラレル I/O	8255	Z80-PIO	6821/6321
シリアル I/O	8251	Z80-SIO	6850/6350
タイマ	8253	Z80-CTC	6840/6340
その他	8237 (DMAC)	Z80-DMA (DMAC)	6845 (CRTC)

13・2 Z80 周辺 LSI の接続

64180 は Z80 上位コンパチブルになっています。ただし、Z80 周辺 LSI とデ
イジーチェーン接続する場合、64180Z を用います。64180R1 を使用する場合ハー
ドウェアに制限があり、外付回路が必要です。64180Z でもリセット直後は 64180
R1 と同じ信号タイミングです。そのため、動作モードコントロールレジスタを
設定して、Z80 用の信号タイミングに変更します。表 13・2 に動作モードコント
ロールレジスタの設定方法を示します。なお、64180 と Z80 の信号名が少し違い
ます。64180 の $\overline{\text{IOE}}$ 、 $\overline{\text{LIR}}$ 信号が Z80 の $\overline{\text{IORQ}}$ 、 $\overline{\text{M1}}$ 信号に相当します。

表 13・2 64180Z での動作モードコントロールレジスタの設定

No.	Z80 周辺 LSI			OMCR へのビットセット		
	デイジーチェーン	CTC	PIO	LIRE	LIRTE	IOC
1	有	有	有	0	"0"をライト*	0
			無	0	—	0
		無	有	0	"0"をライト*	0/1
			無	0	—	0/1
2	無	有	—	1	—	0
		無	—	1	—	0/1

* Z80PIO のレジスタ設定後 1 回実行

Z80 周辺 LSI を使用する場合、割込みの優先順位を決定できるデイジーチェ
イン接続に特徴があります。デイジーチェーンは、Z80 周辺 LSI の $\overline{\text{IEI}}$ 、 $\overline{\text{IEO}}$ 端
子を接続して実現します。図 13・3 では SIO が一番優先順位が高くなっていま
す。デイジーチェーン接続を行うときに注意しなければならないのは、64180Z
の ASCII、DMAC などの内蔵周辺機能とはデイジーチェーン接続できないこと
です。そのため、デイジーチェーン接続の特徴になっている優先順位の高いデバ
イスが割込みルーチンを実行中は、優先順位の低いデバイスは割込みを要求でき
ないという機能が 64180Z 内蔵の周辺機能に対して使えません。また、デイジー
チェーン接続されている Z80 周辺 LSI からの割込みルーチンを実行中に、64180Z
内蔵の周辺機能から割込み要求があり、その割込みルーチンの最後で RETI 命
令を実行すると、Z80 周辺 LSI は自分の割込みルーチンが終了したと勘違いし

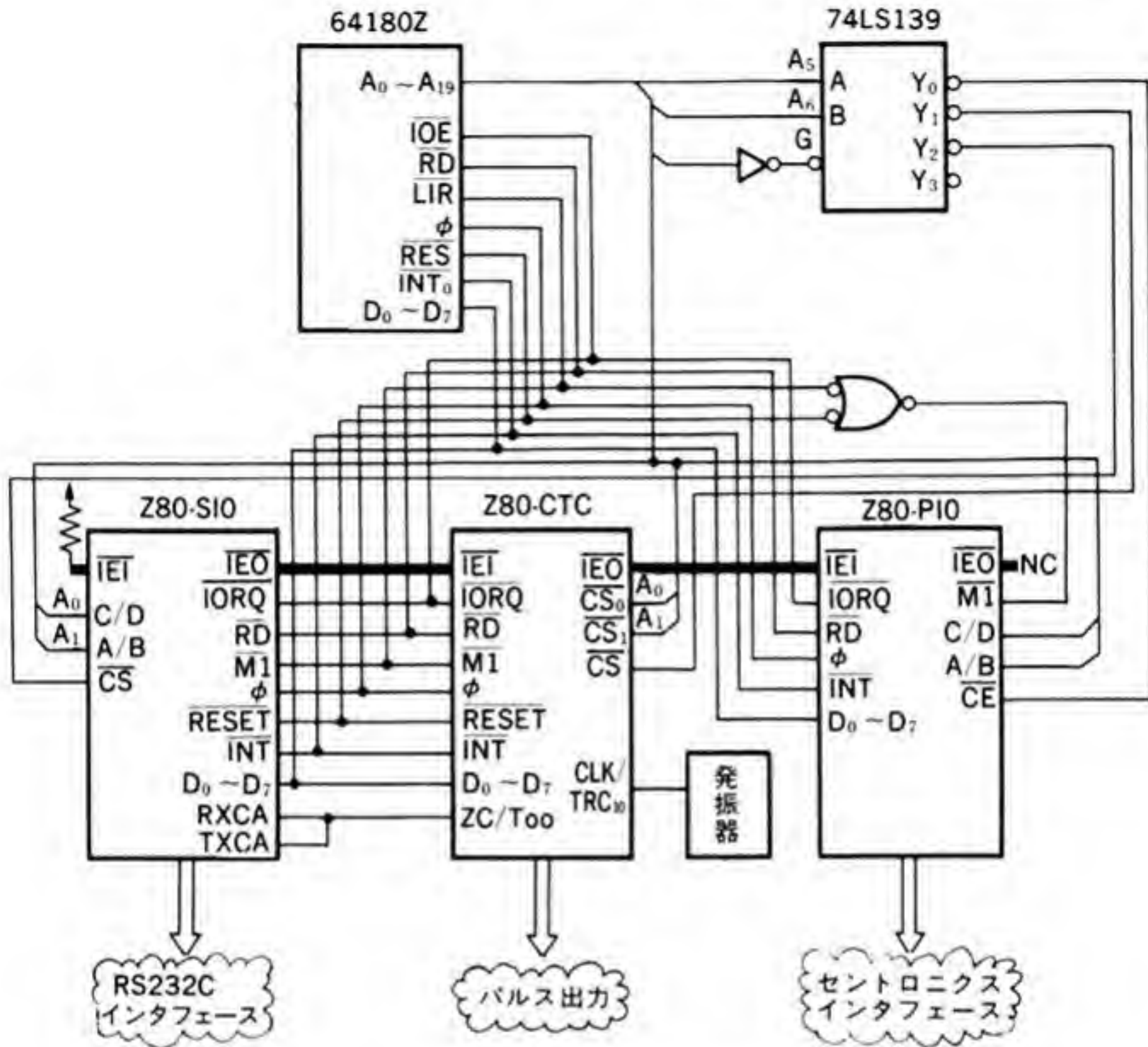


図 13・3 Z80 周辺 LSI 接続例

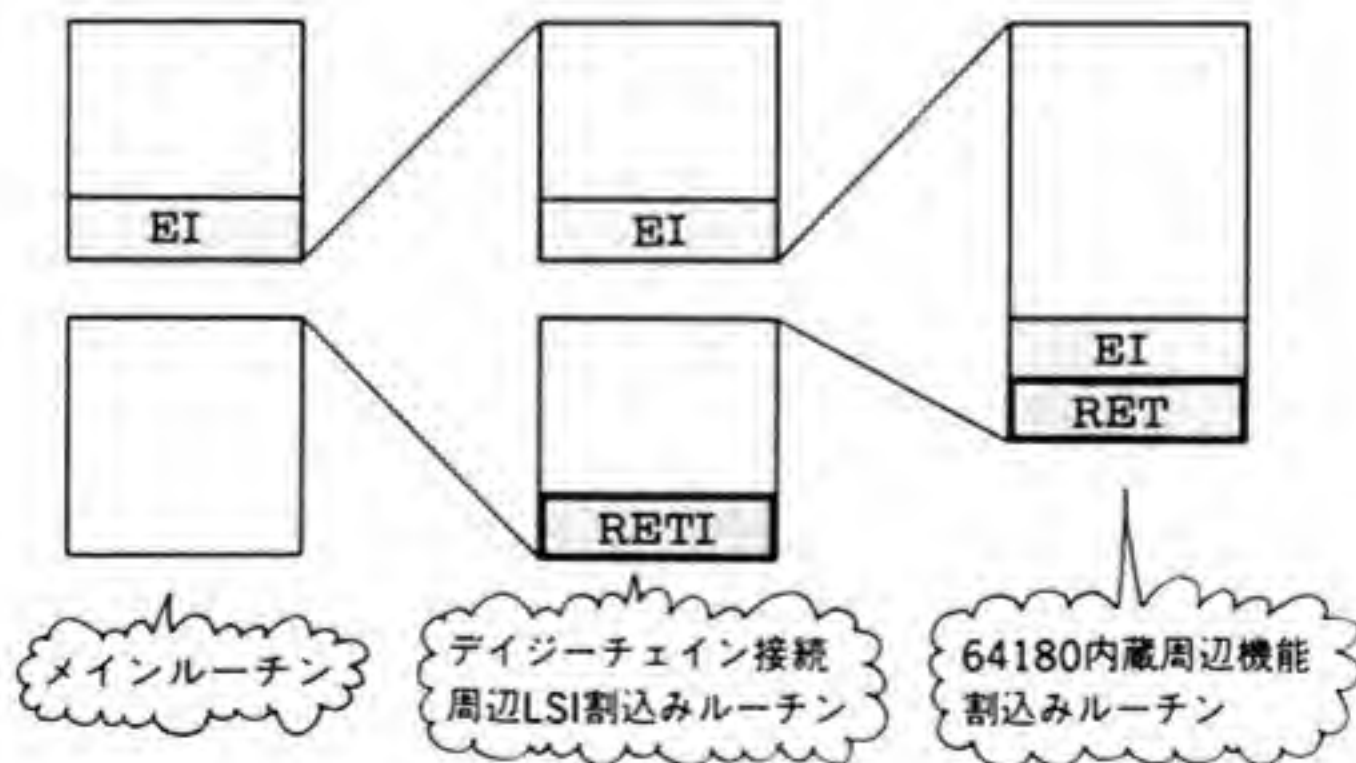


図 13・4 デジチェーン接続時の割込みルーチンからの復帰

13. I/O インタフェース

て、 $\overline{\text{IEO}}$ 信号を 'High' にし優先順位の低い周辺 LSI への割込み禁止を解除します。この勘違いは、デイジーチェーン接続されていない割込みルーチンで RETI 命令を実行したのが原因です。そこで、デイジーチェーン接続を行っている場合、デイジーチェーン接続をしていない 64180Z 内蔵の周辺機能や他の周辺 LSI の割込みから復帰するときは、図 13・4 のように RET 命令を使用します。

通常、64180 ではデイジーチェーン接続を行わないほうが、システム構築が考えやすくなります。

13・3 68系周辺LSIとの接続

64180は68系周辺LSI用にE信号を出力しています。E信号を使えば、68系周辺LSIを80系CPUの64180にダイレクトに接続できます。図13・5にHD63B21を接続した例を示します。68系周辺LSIを接続する場合に注意しなければならないのは、E信号の‘High’出力期間です。64180のバスは非同期バスであるため、メモリ、I/Oのリード・ライト間隔が一定ではありません。そのため、E信号の波形が不規則になります。表13・3に、64180の各状態におけるE信号の出力タイミングを示します。E信号の‘High’期間が最少幅になるのは、I/Oライトサイクル時です。たとえば、HD63B21のE信号‘High’幅は最低220 ns以上必要です。そのため、64180の動作周波数は $\phi=4.5$ MHz以下で使用するか、ウェイトサイクルを挿入します。

また、E信号の出力形がduty 50%でないため、周辺LSIがE信号を基にして外部とインタフェースしている場合、注意が必要です。たとえば、HD63B21では、ハンドシェイク制御出力から64180のEクロックサイクル幅がそのまま出

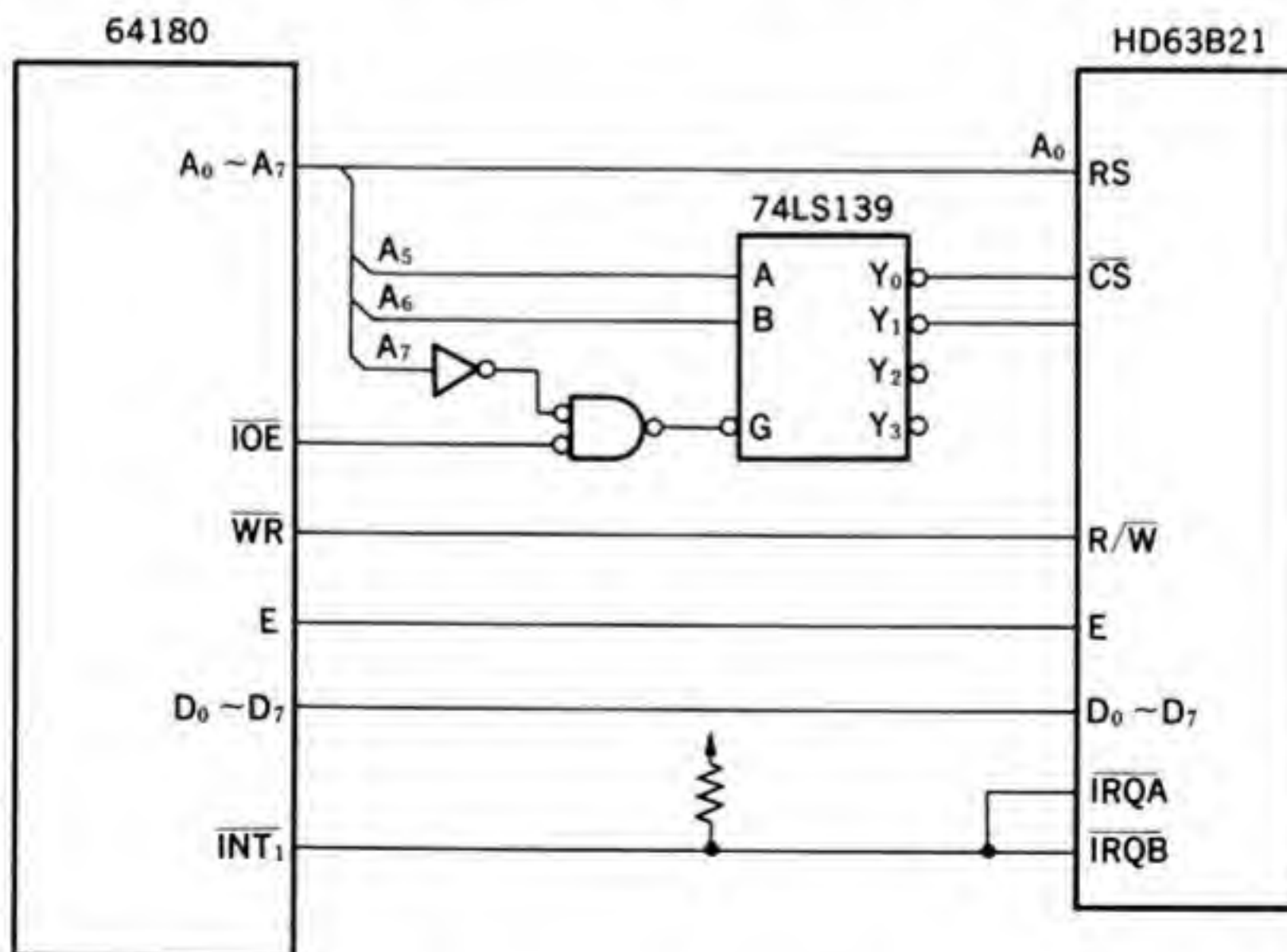


図13・5 68系周辺LSIの接続例

13. I/O インタフェース

表 13・3 各状態における E 信号の出力タイミング

バ ス 状 態	E 信号の 'High' タイミング	E 信号の 'High' 期間
オペコードフェッチ メモリリード/ライトサイクル	$T_2 \nearrow \sim T_3 \searrow$	$1.5\phi + n\phi$
I/O リードサイクル	1 st $T_W \nearrow \sim T_3 \searrow$	$0.5\phi + n\phi$
I/O ライトサイクル	1 st $T_W \nearrow \sim T_3 \nearrow$	$n \cdot \phi$
NMI アクノレッジ第 1 マシンサイクル	$T_2 \nearrow \sim T_3 \searrow$	1.5ϕ
INT ₀ , INT ₁ , INT ₂ 内部割込み アクノレッジ第 1 マシンサイクル	1 st $T_W \nearrow \sim T_3 \searrow$	$0.5\phi + n\phi$
Bus Release, Sleep System Stop モード	$\phi \searrow \sim \phi \searrow$	2ϕ または 1ϕ

n : WAIT ステートの挿入数

力されます。ところが、同じ 68 系周辺 LSI である 6845 は内部レジスタのアクセスのみに E 信号を使用しているため、E 信号の 'High' 期間のみ注意すればいいことになります。

割込みを使う場合は、HD63B21 の割込み出力信号はオープンドレイン出力となっているため、プルアップ抵抗を接続し、レベルセンス、ベクタ方式の外部割込み信号の $\overline{\text{INT}}_1$ 、または $\overline{\text{INT}}_2$ に接続すればいいでしょう。

13・4 FDC の 接 続

64180 に FDC (Floppy Disk Controller) を接続する場合、64180 と FDC 間のデータ転送方法として、**プログラム転送**を使う方法と **DMA 転送**を使う方法があります。通常、他の CPU ではプログラム転送を使うため、他のプログラムを実行することができません。また、DMA 転送を使うと高価な DMAC が必要になります。それに対して、64180 は DMAC を内蔵しているため、外付 DMAC なしで DMA 転送を行うことができ、プログラム実行のスループットを上げることができます。図 13・6 に、64180 と FDC の HD63265 の接続例を示します。

HD63265 は高精度データセパレータ回路を内蔵しているため、外付 VFO 回路が不要であり、標準の FDC とソフトウェアコンパチであり、また、80 系、68 系両バスインタフェースを備えた FDC です。HD63265 の DMA 制御信号の $\overline{\text{DREQ}}$ 、 $\overline{\text{TEND}}$ 信号は、64180 の $\overline{\text{DREQ}}_1$ 、 $\overline{\text{TEND}}_1$ 信号に接続します。なお、 $\overline{\text{DACK}}$ 信号に対応する信号を 64180 はもっていません。なぜなら、64180 内蔵の DMAC は、メモリアドレスと I/O アドレスを別々に出力するデュアルアドレス方式の DMA 転送をサポートしているからです。そのため、シングルアドレス方式の DMA 転送

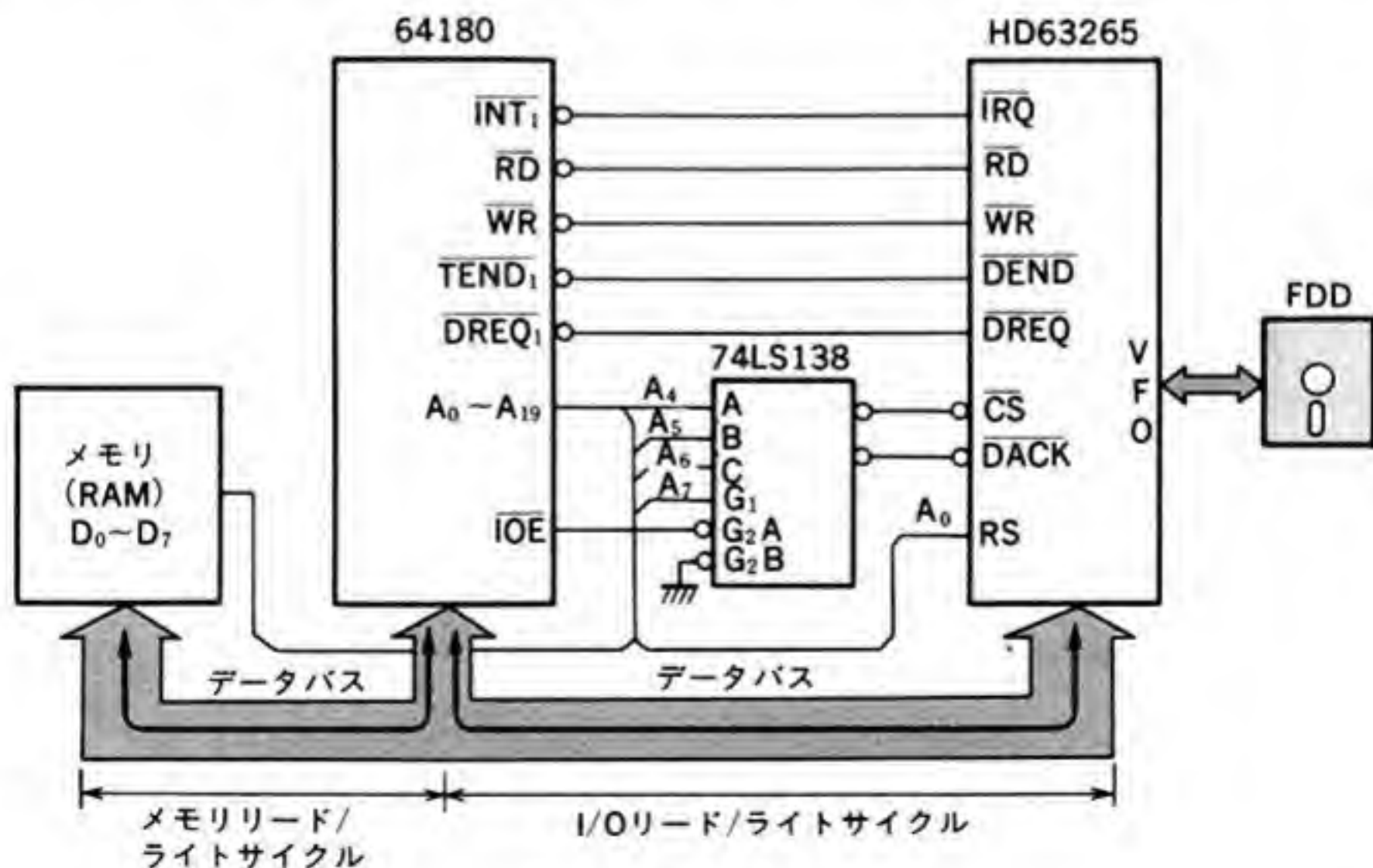


図 13・6 FDC の接続例

13. I/O インタフェース

表 13・4 FDC アドレスマップ

I/O アドレス	内 容
80H	FDC (HD63265) ステータスレジスタ
81H	FDC (HD63265) データレジスタ
90H	FDC (HD63265) DACK 信号用 (DMA 時)

用の FDC と接続する場合、 $\overline{\text{DACK}}$ 信号を別の方法で生成します。64180 は I/O のリード/ライト時と同様に、DMA 転送時も I/O アドレスを出力します。そこで、DMA 転送用の I/O アドレスを設定しておき、 $\overline{\text{CS}}$ 信号と同様にデコード信号を $\overline{\text{DACK}}$ 信号に入力します(表 13・4)。図 13・7 に、64180 と HD63265 の DMA 転送タイミングを示します。

$\overline{\text{DREQ}}_1$ 信号はレベル入力とエッジ入力を選択可能ですが、HD63265 とのインタフェースではエッジ入力を選択します。DMA ライトサイクル時 $\overline{\text{DACK}}$ 信号が 'Low' になると、HD63265 は $\overline{\text{DREQ}}$ 信号を 'High' にします。64180 は T_3 ステートの前の、ステートの ϕ の立上りで $\overline{\text{DREQ}}_1$ 信号をサンプリングしているため、レベル入力を選択すると $\overline{\text{DREQ}}_1$ 信号が 'High' になる前にサンプリングしてしまい、DMA 転送が余分に行われます。エッジ入力を選択しておくで、立下りエッジを検出しないと DMA 転送を行わないため、 $\overline{\text{DREQ}}$ 信号の立上りが遅れてもだいじょうぶです。

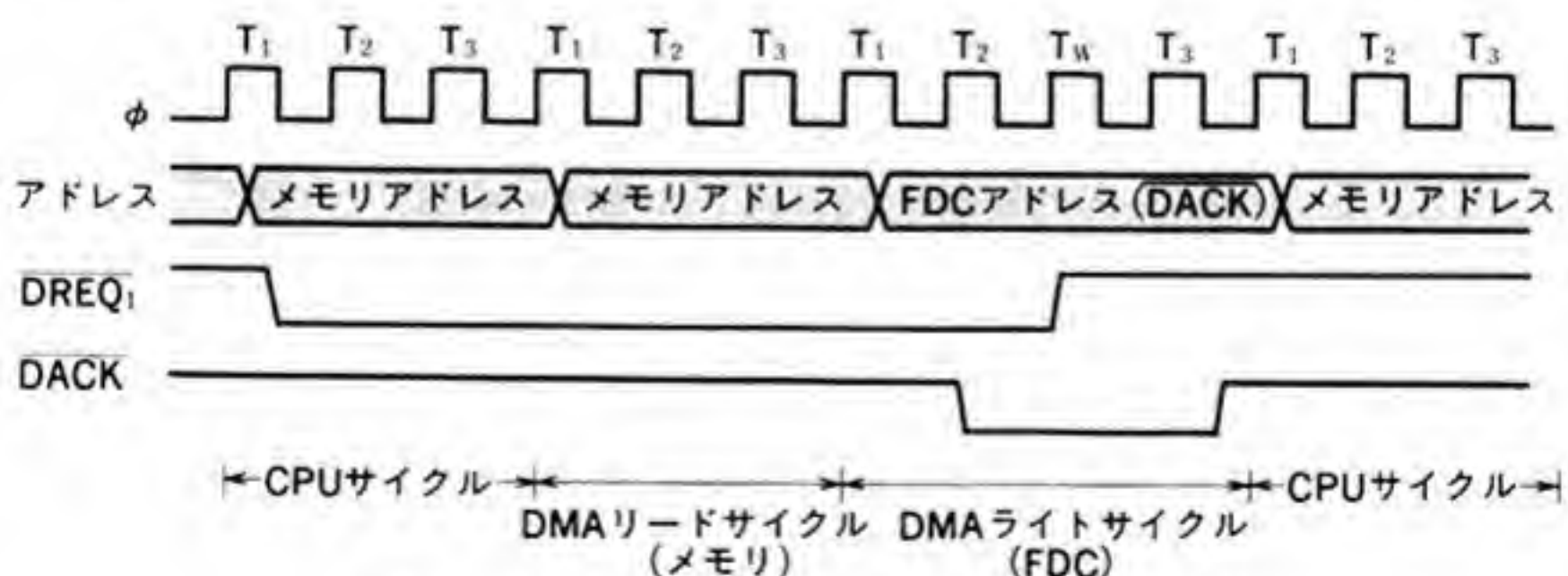


図 13・7 メモリ→FDC データ転送タイミング

13・5 CRTC の 接 続

64180 を使用して CP/M システムなどのパソコンやワープロを構築する際必要になるのが、CRT コントローラ（以下 CRTC と略す）や LCD タイミングコントローラ（以下 LCTC と略す）などの表示系コントローラの制御です。HD 6445 は CRT 表示に必要な各種タイミング信号を発生し、画面分割やスムーズスクロールなどの各種機能を備えた、小形の簡易キャラクタディスプレイから高解像度のフルグラフィックシステムまで適用できる CRTC です。図 13・8 に、HD 6445 を使用したキャラクタ表示システムを示します。

64180 と HD6445 を接続する場合、 \overline{CS} 信号はアドレスのみから生成します。

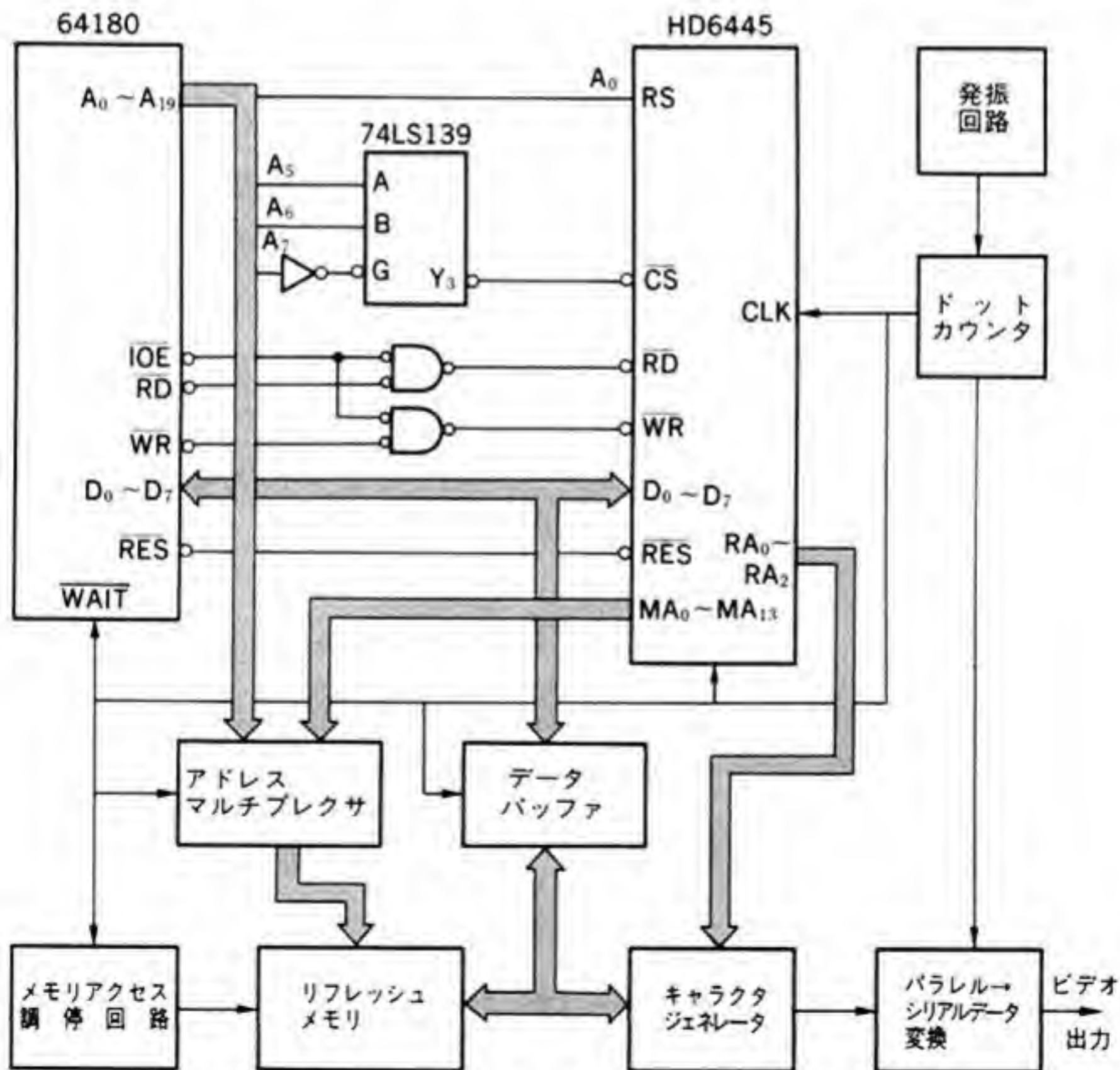


図 13・8 HD6445 による CRT 表示システム構成例

13. I/O インタフェース

HD6445 の \overline{RD} , \overline{WR} 信号は 64180 の \overline{IOE} 信号と \overline{RD} , \overline{WR} 信号の AND をとって生成します。HD6445 は表示タイミングのみ出力するため、64180 が表示データをリフレッシュメモリにライトし、その表示データを HD6445 が CRT に出力する形をとります。リフレッシュメモリを 64180 と HD6445 の 2 つの LSI がアクセスするため、バスの競合がおこります。バスの競合を回避するため、64180 と HD6445 の出力アドレスを切り替えるアドレスマルチプレクサ回路、データバスを切り替えるデータバッファ、およびアクセスタイミングを調整するメモリアクセス調停回路が必要になります。

リフレッシュメモリへのアクセスを CPU 優先にすると、CRTC アクセスが阻害されるため、CRT 上にちらつきが出ます。ちらつきを防止するためには、CRTC のアクセスを優先する必要があります。CRTC アクセスを優先するために、CRTC がリフレッシュメモリをアクセスしている間、CPU がリフレッシュメモリをアクセスしないように、ハードウェアまたはソフトウェアで対処します。もともと非同期である CRTC と CPU のバスタイミングを、CPU がリフレッシュメモリをアクセスするときのみ 64180 の \overline{WAIT} 信号を制御して、CRTC のバスタイミングにあわせます。この方法はメモリアクセス調停回路が複雑になりますが、プログラム実行のスループットが上がります。図 13・9 にアクセスタイミングを示します。

高解像度グラフィックシステム、たとえば 640×400 ドット RGB カラー表示システムでは、96 K バイトのリフレッシュメモリが必要ですが、64180 内蔵の MMU を使うことで簡単にメモリ管理が行えます。

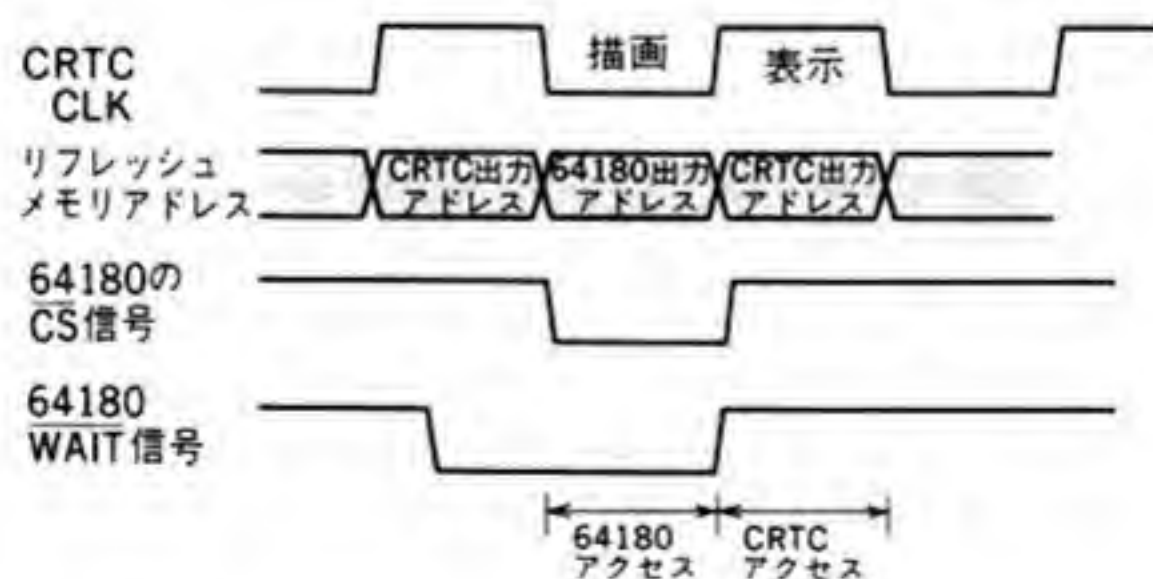


図 13・9 リフレッシュメモリアクセスタイミング

13・6 LCTC の 接 続

13・5 節では CRTC を接続しましたが、本節では LCD タイミングコントローラ（以下 LCTC と略す）を接続した例を示します（図 13・10）。HD64645 は、大容

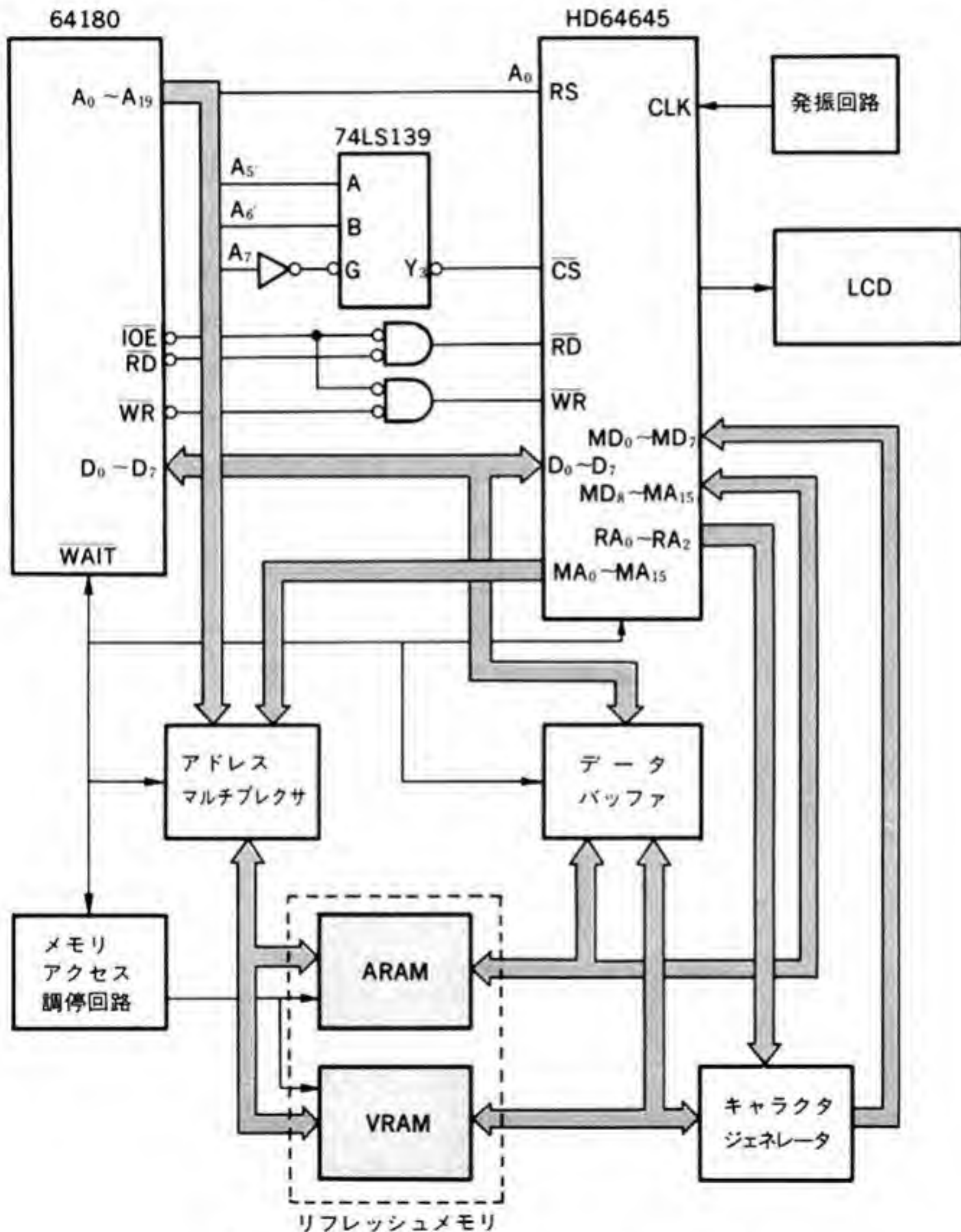


図 13・10 HD64645 による LCD 表示システム構成例

13. I/O インタフェース

量ドットマトリクス液晶表示コントローラで、6845CRTC とソフトウェアコンパチブルであり、さらにスムーズスクロール、キャラクタ単位の反転ブリンク、キャラクタ・グラフィック重合せ表示用のアトリビュート機能をもっています。そのため、6845 を使用した CRT 表示システムを LCD 表示システムに置き替える場合、ソフトの遺産を活用できます。

HD64645 による LCD 表示システムは、基本的に HD6445 による CRT 表示システムと同様です。64180 と HD64645 のバスの競合がおきないように、64180 と HD64645 の出力アドレスを切り替えるアドレスマルチプレクサ回路、データバスを切り替えるデータバッファ、アクセスタイミングを調整するメモリアクセス調停回路が必要です。HD64645 は、リフレッシュメモリを 2 バイト単位で管理しています。たとえば、キャラクタ表示モードのときはキャラクタデータ用 VRAM とアトリビュート用 ARAM の 2 バイト構成で、HD64645 は 2 バイト同時にアクセスします。64180 はリフレッシュメモリを 1 バイト単位でアクセスするため、VRAM と ARAM のアドレスを違える必要があります。

また、CRTC の場合と同様に、リフレッシュメモリへのアクセスを LCTC 優先にする必要があります。LCTC がリフレッシュメモリをアクセスしている間、CPU がリフレッシュメモリをアクセスしないように、ハードウェアまたはソフトウェアで対処します。もともと非同期である、LCTC と CPU のバスタイミングを CPU がリフレッシュメモリをアクセスするときのみ 64180 の WAIT 信号を制御して、LCTC のバスタイミングにあわせます。この方法は、メモリアクセス調停回路が複雑になりますが、少しの待ち時間でリフレッシュメモリをアクセスできるため、プログラム実行のスループットが上がります。

14. 新規命令

64180 は Z80 の 158 個の命令に加え、7 種類 12 個の命令が追加され 165 個の命令をもっています。これらの命令は、64180 に内蔵された周辺機能を効率よく使用したり、Z80 に比べ大幅にスループットを向上させるうえで不可欠な命令です。また、スリープ命令のように CMOS の低消費電力を生かすうえで、重要な命令も含まれています。これらの命令を使用することにより、効率よいプログラムを作成することができます。

14・1 新規命令の概要

64180 は、Z80 と互換性のある命令セット (158 命令) に加えて、7 種類 12 個の命令が追加され 165 個から成る命令セットをもっています。(表 14・1)。

表 14・1 追加命令一覧

	ニーモニック	区 分	機 能
1	INO g, (m)	I/O 命令	$(OOm)_1 \rightarrow g$ $m \rightarrow A_0 \sim A_7$ $OO \rightarrow A_8 \sim A_{15}$
2	OUTO (m), g	I/O 命令	$g \rightarrow (OOm)_1$ $m \rightarrow A_0 \sim A_7$ $OO \rightarrow A_8 \sim A_{15}$
3	OTIM	I/O 命令	$(HL)_M \rightarrow (OOC)_1$ $HL + 1 \rightarrow HL$ $C + 1 \rightarrow C \quad C \rightarrow A_0 \sim A_7$ $B - 1 \rightarrow B \quad OO \rightarrow A_8 \sim A_{15}$
	OTIMR		$Q \left[\begin{array}{l} (HL)_M \rightarrow (OOC)_1 \\ HL + 1 \rightarrow HL \\ C + 1 \rightarrow C \quad C \rightarrow A_0 \sim A_7 \\ B - 1 \rightarrow B \quad OO \rightarrow A_8 \sim A_{15} \end{array} \right.$ Repeat Q Until B=0
	OTDM		$(HL)_M \rightarrow (OOC)_1$ $HL - 1 \rightarrow HL$ $C - 1 \rightarrow C \quad C \rightarrow A_0 \sim A_7$ $B - 1 \rightarrow C \quad OO \rightarrow A_8 \sim A_{15}$
	OTDMR		$Q \left[\begin{array}{l} (HL)_M \rightarrow (OOC)_1 \\ HL - 1 \rightarrow HL \\ C - 1 \rightarrow C \quad C \rightarrow A_0 \sim A_7 \\ B - 1 \rightarrow B \quad OO \rightarrow A_8 \sim A_{15} \end{array} \right.$ Repeat Q Until B=0
4	MLT WW	8ビット算術命令	$WWHr \times WWLr \rightarrow WW_R$
5	TSTIO m	8ビット論理演算命令	$(OOC)_1 \cdot m \quad C \rightarrow A_0 \sim A_7$ $OO \rightarrow A_8 \sim A_{15}$
6	TST g	8ビット論理演算命令	$A_{cc} \cdot g$
	TST (HL)		$A_{cc} \cdot (HL)_M$
	TST m		$A_{cc} \cdot m$
7	SLP	CPUコントロール	Sleep

この追加された 12 命令は、次の 3 種類に大別することができます。

(1) 64180 の内蔵する周辺機能を効率よく使用するための入出力命令

INO, OUTO 命令など

(2) Z80 に備えていない命令で処理能力を向上する命令

① 論理演算命令、**TST 命令**など

② 乗算命令 (**MLT 命令**)

(3) CMOS の低消費電力を有効に活用するための命令 (**SLP 命令**)

(1) の INO, OUTO などの命令は、内蔵の周辺機能を使用する場合に便利です。内蔵の周辺機能のレジスタは、64180 のもつ 64 K バイトの I/O 空間中の 64 バイトに割り当てられており、INO, OUTO などの命令は、この周辺機能のレジスタを効率よくアクセスするための入出力命令です。

(2) の TST 命令は、RAM 上のフラグをチェックする場合などに使用します。TST 命令は AND 命令とは異なり、RAM や I/O レジスタの値に影響を与えることなくフラグなどのチェックが可能です。また、MLT 命令は制御応用には必須の機能で、8 ビット×8 ビットの乗算命令です。(2) の TST 命令、MLT 命令はいずれも Z80 がもっていない機能を命令としてもち、実行ステートの低減を実現する命令です。

(3) は低消費電力モード (スリープモード) を設定する命令です。SLP 命令は、64180 のような CMOS のマイクロプロセッサには必須の命令といえます。

また、これらの拡張命令は、マクロ 80 のような従来、8085, Z80 用として広く普及しているマクロアセンブラを用いて使用することができます。具体的には、拡張された命令をマクロ定義することにより、Z80 用のマクロアセンブラを 64180 用のアセンブラとして使用することができます。

14・2 入出力命令の使用例

64180 は内蔵の周辺機能を効率よく使用するため、入出力命令 (I/O 命令) を追加しています。追加された命令は、次に示す 3 種類 6 つの命令です。

- (1) INO 命令
- (2) OUTO 命令
- (3) OTIM, OTIMR, OTDM, OTDMR 命令

内蔵周辺機能のレジスタは、64 K バイトの I/O 空間上の 0 番地から 256 番地の中に配置されています。実際には、この中の 64 バイトだけが使用されています。Z80 の命令を用いてこれらのレジスタをアクセスしようとする場合、ロード命令で BC レジスタに 16 ビットのアドレスを設定したうえで、OUT (C), g 命令などの命令を使用します。従来、Z80 で I/O の書込みに使用した OUT (m), A は、アキュムレータの内蔵がアドレスの上位 ($A_8 \sim A_{15}$) に出力されるため、内蔵周辺のレジスタ書込みにには使用できません (図 14・1)。

これに対し (1)~(3) の命令は、BC レジスタへの設定をせずに周辺機能のレジスタをアクセス可能です。(1)~(3) の追加命令は、オペランドで下位 8 ビット ($A_7 \sim A_0$) を設定し、上位 8 ビット ($A_{15} \sim A_8$) には自動的に OOH が設定されます。

たとえば、内蔵の DMAC のレジスタを設定する例を示します。DMAC は設定すべきレジスタが多く、効率よくレジスタの設定を行う必要があります。Z80 では、OUTI, OUT (C), A 命令を使用しています。これに対し 64180 では、OTIMR, OUTO 命令を使用し、レジスタの設定を行っています。ここで OTIMR

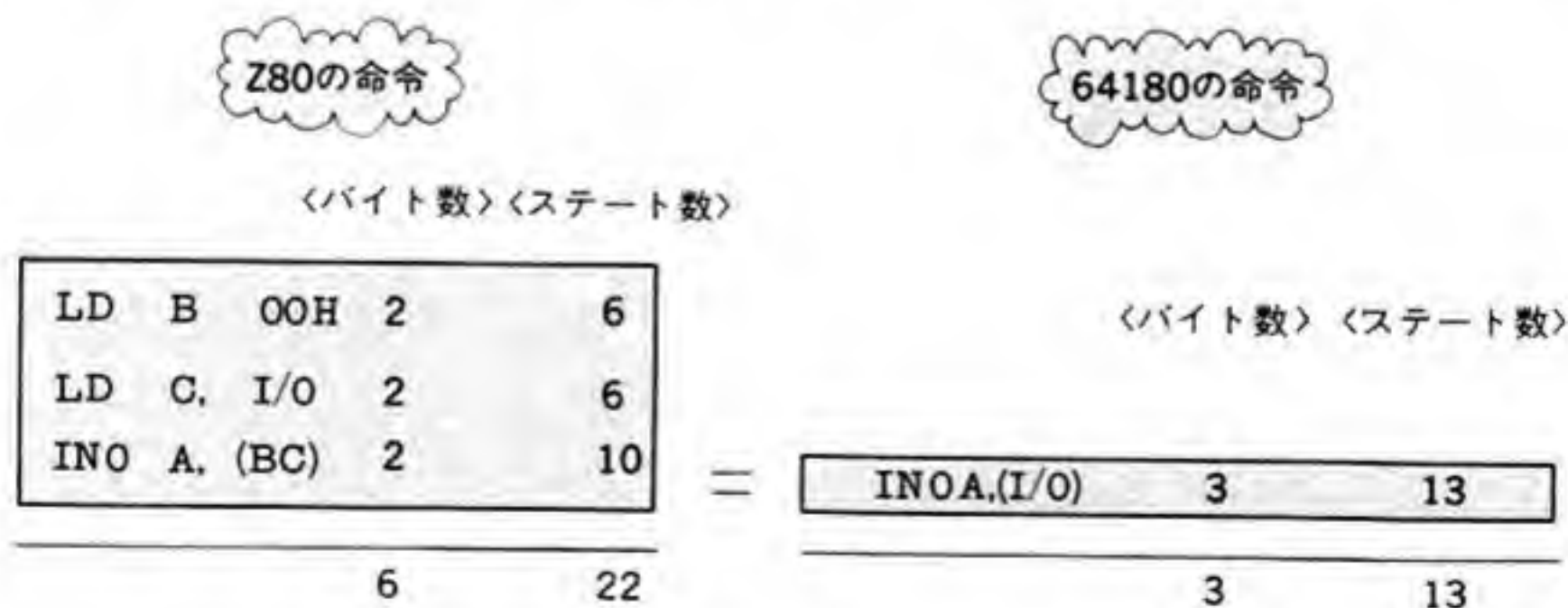


図 14・1 I/O 空間上のレジスタ設定を行うプログラム例

Z80の命令を使用した場合

64180の命令を使用した場合

<バイト数> <ステート数>

<バイト数> <ステート数>

	LD	HL,1000H	3	9		LD	HL,1000H	3	9
	LD	BC,0020H	3	9		LD	BC,0820H	3	9
	LD	D,08H	2	6					
LOOP	OUTI		2	12		OTIMR		2	16 不成立 14 成立
	INC	B	1	4					SAR DARの設定 BCR
	INC	C	1	4					
	DEC	D	1	4					
	JR	NZ,LOOP	2	6 不成立 8 成立					
	LD	A,C3H	2	6		LD	A,C3H	2	6
	LD	C,31H	2	6					
	OUT	(C),A	3	10		OUTO	A,31H	2	13 DMAモード レジスタの設定
	LD	A,66H	2	6		LD	A,66H	2	6
	DEC	C	1	4					
	OUT	(C),A	3	10		OUTO	A,30H	2	13 DMAステータス レジスタの設定 (DMAイネーブル)
14行				28	320	7行			
						16			
						182			

図 14・2 追加された入出力命令の使用例

命令はI/Oへのブロック転送命令で、メモリから連続するI/Oへのブロック転送を行います。この例では、OUTO、OTIMR命令の使用により、プログラムライン数、バイト数、実行ステート数とも、Z80の命令を使用した場合に比べ50～60%に削減しています(図14・2)。

なお、このプログラムはチャンネル0のDMACで、メモリ↔メモリの転送をサイクルスチールモードで実行させる例です。

14・3 論理演算命令 (TSTIO命令)の使用例

TSTIO 命令も、主に 64180 の内蔵する周辺機能を考慮して追加された命令です。TSTIO 命令は Cレジスタに下位 8 ビットの I/O アドレス ($A_7 \sim A_0$) を指定すると上位 8 ビット ($A_{15} \sim A_8$) に '00' が指定され、このアドレスのレジスタの値とオペランドで指定した 8 ビットのデータとの論理積をとる命令です。結果はフラグにのみ反映される非破壊の論理演算を行います (図 14・3、表 14・2)。

Z80 の命令を使用する場合、IN A, (C) などの入力命令により、アキュムレータに I/O のレジスタを読み出し、論理演算命令 (AND m など) により '1'、'0' の判定を行います。64180 では、この動作を TSTIO 命令、1 命令で実現することができます。また AND 命令の場合、1 度論理演算を実行するとその値が破壊されるため、再度論理演算を行う場合、もう 1 度 I/O から読み出すか他のレジスタに退避しておく必要があります。これに対して、TSTIO 命令は非破壊であるため、何度でも異なる値との論理演算を実行することができます。

たとえば、TSTIO 命令を用いて内蔵の非同期シリアル (ASCI) のフラグを

Z80の命令

〈バイト数〉 〈ステート数〉

LD C, I/OH 2 6

LD B, 00H 2 6

IN A, (BC) 2 9

AND m 2 6

=

JR Z, LOO 2 { 6 不成立
8 成立

計 10 35

64180の命令

〈バイト数〉 〈ステート数〉

LD C I/O 2 6

TSTIO m 3 12

JR Z, LOO 2 { 6 不成立
8 成立

計 7 26

図 14・3 TSTIO 命令の等価機能

表 14・2 TSTIO 命令と AND 命令の実行結果

	フ ラ グ	レジスタへの反映
TSTIO 命令	変化する	反映しない (非破壊)
AND 命令	変化する	A _{cc} に反映する (破壊)

Z80の命令を使用した場合				64180の命令を使用した場合			
		〈バイト数〉 〈ステート数〉				〈バイト数〉 〈ステート数〉	
LD	C,04H	2	6	LD	C 04H	2	6
LD	B,00H	2	6				
IN	A,(C)	2	9				
LD	B,A	1	4				
LD	A,74H	2	6				
AND	B	1	4	TSTIO	74H	3	12 RDRFのみ1
JR	Z,LO1	2	6 不成立 8 成立	JR	Z,LO1	2	6 不成立 8 成立
LD	A,34H	2	6				
AND	B	1	4	TSTIO	34H	3	12 RDRF=1 OVRN=1
JR	Z,Lφ2	2	6 不成立 8 成立	JR	Z,LO2	2	6 不成立 8 成立
LD	A,04H	2	6				
AND	B	1	4	TSTIO	04H	3	12 RDRF(PE+FE)=1
JR	Z,LO3	2	6 不成立 8 成立	JR	Z,LO3	2	6 不成立 8 成立
JR	DCD	2	8	JR	DCD	2	8
14行	24	81		8行	19	68	

図 14・4 TSTIO 命令の使用例

判定し、フラグに応じた処理を行うプログラムを考えてみましょう。非同期シリアルステータスレジスタ（チャンネル0はI/O空間の04H、チャンネル1は05H）には、RDRF、OVRNなど5つ（チャンネル0の場合DCDも加え6つ）のステータスフラグがあります。ここでは受信のみについて考慮すると、RDRF、OVRN、PE、FE、DCDの計5つのフラグを判定する必要があります。64180はZ80に比べ、TSTIO命令により効率よくフラグの判定を行えるため、Z80の命令のみを使用した場合に比べ、ライン数で60%、バイト数、実行サイクル数でも80%程度にプログラムを削減することができます（図14・4）。

なお、ここに示したプログラムでは、簡単にするためパリティエラー（PE）とフレーミングエラー（FE）に関する処理は、同一のプログラム（プログラム中のLO3の処理）としてあります。

14・4 MLT命令の使用例

64180 は、8ビット×8ビットの乗算命令をもっています。この乗算命令は、BCレジスタなどのレジスタペアを使用して8ビット×8ビットの乗算を行います。たとえば、Bレジスタ、Cレジスタにそれぞれ乗数、被乗数を設定しMLT命令を実行すると、積（答え）がBCのレジスタペアにストアされます。この16ビットのペアレジスタとしては、BCレジスタ、DEレジスタ、HLレジスタ、スタックポインタ（SP）の4種類のレジスタを使用できます。MLT命令の実行には17ステートかかり、実際の実行時間は8MHz版で2.1μsかかります。

従来、Z80では乗算命令をもっていなかったため、シフト命令を使用して乗算のプログラムを作成する必要がありました。これに対し64180は、MLT命令に

Z80の命令を使用した場合				64180の命令を使用した場合			
		〈バイト数〉	〈ステート数〉			〈バイト数〉	〈ステート数〉
LD B, 08H		2	6				
LD A, (RAMB)		3	12	LD A, (RAMB)	3	12	
LD (RSLT+1), A		3	13	LD H, A	1	4	
XOR A		1	4	LD A, (RAMA)	3	12	
LD (RSLT), A		3	13	LD L, A	1	4	
LOO	LD HL, RSLT	3	9	MLT HL	2	17	
SLA (HL)		2	13				
INC HL		1	4				
RL (HL)		2	13				
JR NC, LO1		2	6 不成立 8 成立				
LD HL, (RAMA)		3	15				
LD DE, (RSLT)		4	18				
ADD HL, DE		1	7				
LD (RSLT), HL		3	16	LD (RSLT), HL	3	16	
LO1	DEC B	1	4				
JR NZ, LOO		2	6 不成立 8 成立				
16行		36	938	6行		15	65

図 14・5 MLT命令の使用例

表 14・3 追加命令によるスループットの向上

	ライン数		64180 /Z80	バイト数		64180 /Z80	ステート数		64180 /Z80
	Z80の 命令	64180 の命令		Z80の 命令	64180 の命令		Z80の 命令	64180 の命令	
入出力命令	14	7	0.5	28	16	0.62	320	182	0.75
TSTIO命令	14	8	0.57	24	19	0.79	81	68	0.84
MLT 命令	16	6	0.38	36	15	0.42	938	65	0.07
計	44	21	0.48	88	50	0.57	1339	315	0.24

より大幅にスループットの向上を実現することができます(図14・5, 表14・3)。

たとえば, 8ビット×8ビットの乗算を実行しようとした場合, Z80では, シフト命令を使用しループを構成します。一方, 64180ではこれをMLT命令で実現しています。Z80と64180の実際のプログラムを比較すると, 64180ではMLT命令の使用によりプログラムのライン数, バイト数では40%程度, 実行ステート数では10%程度にプログラムを削減しています(図14・6)。

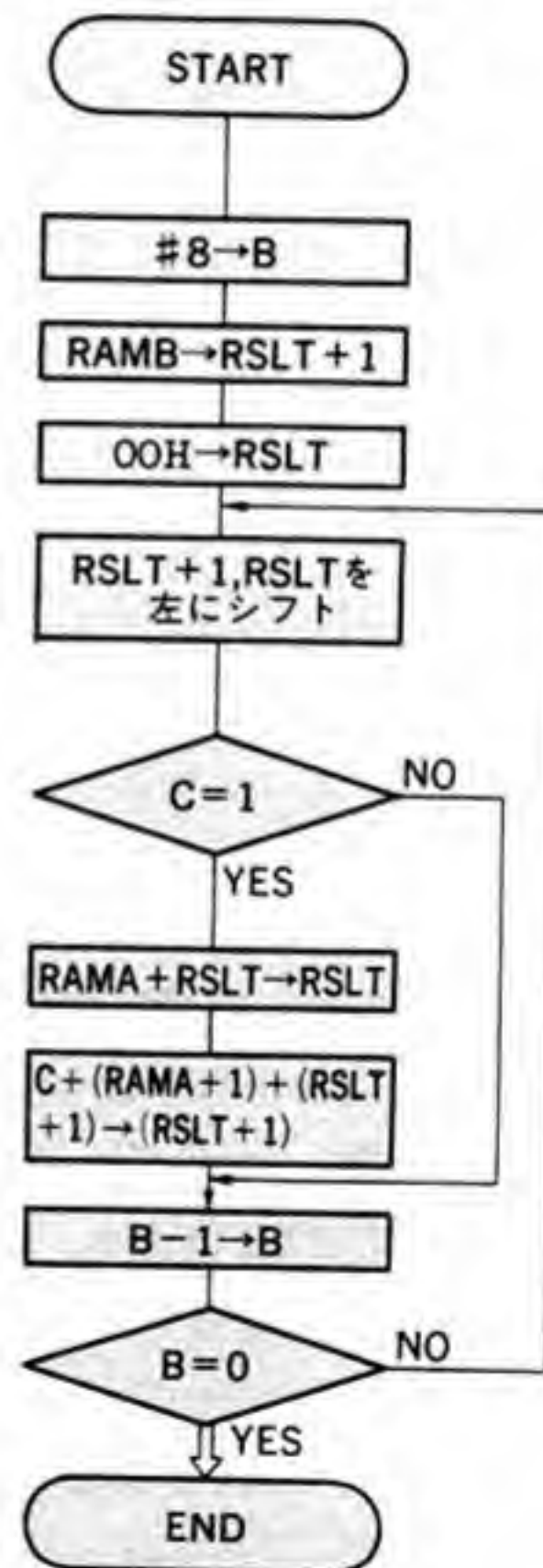


図 14・6 Z80の命令を使用した場合の乗算-フローチャート

拡張命令によるスループットの向上

64180 では追加された命令を使用することにより、Z80 に比べ大幅にプログラムのライン数、バイト数、実行ステート数を削減することができます。表に入出力命令、TSTIO 命令、MLT 命令の使用効果をまとめてみます。表 14・3 は Z80 と 64180 の命令で、入出力、フラグ判定、および 8 ビット×8 ビットの乗算を行った場合のライン数、バイト数、ステート数および Z80 と 64180 の性能比を示してあります。64180 は Z80 に比べ、ライン数で 48 % 程度、バイト数で 57 % 程度、実行ステート数で 24 % 程度にプログラムを削減し、スループットの向上を実現できることがわかります。

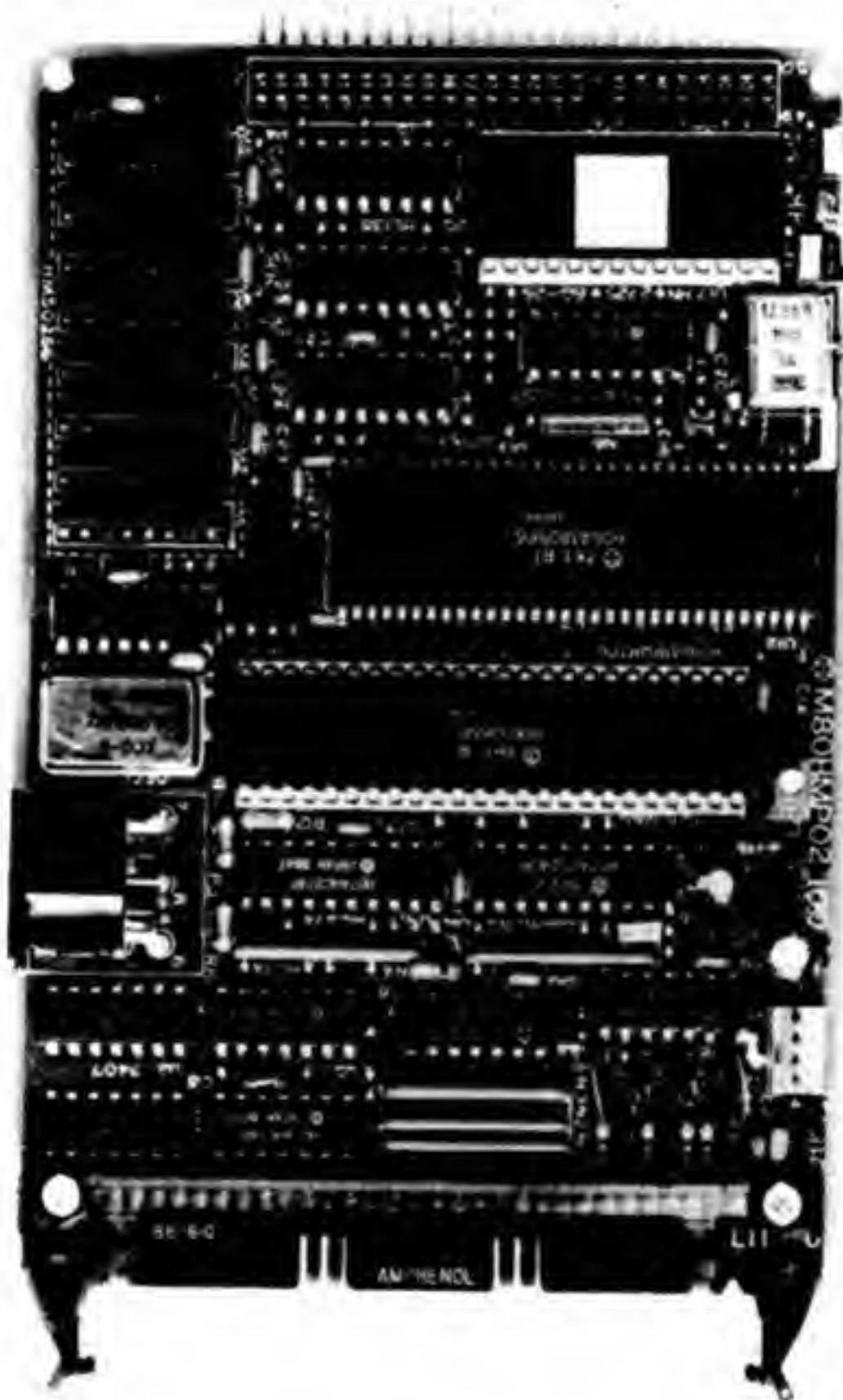
15. 180 カード パソコン

カードパソコンは 80 系ソフトウェア資源を生かしつつ、小形でコンパクトなパソコン機能をもつシステムです。ボードの構成は、80 系の汎用 OS である CP/M Plus が走る最小構成をとっており、CP/M の応用プログラムによるソフトウェア開発装置としても利用できます。主な構成は、メモリ（RAM：256 K バイト、ROM：32 K バイト）、RS-232C 通信インタフェース、プリンタインタフェースおよびフロッピーディスクインタフェースから成り、ハガキサイズ（148×100 mm）の大きさで実現しています。

15・1 カードパソコンについて

64180 を搭載するシステムを構築する場合

- (1) Z80 とその周辺 LSI を用いたシステムの小形化が図れる
- (2) メモリ空間が広くとれ（最大1 M バイト）、80 系オペレーティングシステム（OS）との組合せて高級言語など幅広いソフトウェアの活用ができる



ROM ゴースト

ROM の選択は 64180 のすべてのアドレスについてデュードをしていないために、見かけ上何もないメモリ空間でも ROM が選ばれ、データが読み出される。

図 15・1 外観図

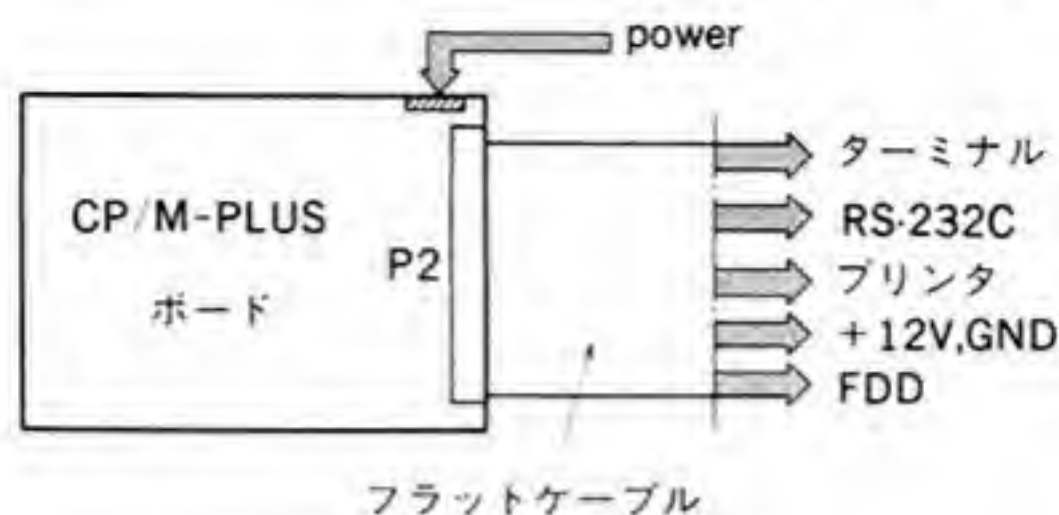


図 15・2 周辺機器接続方法

などの特長を出すことができます。

カードパソコンでは、これらの特長をもつパソコンシステムを実現しています。設計上のコンセプトとして、パソコンをハガキサイズ（148×100 mm）の基板に1ボード化することをハードウェアの主眼とし、さらに膨大な80系ソフトウェア資産を生かせるように、80系標準OSである米国ディジタル・リサーチ社のCP/Mを搭載することをソフトウェア上のポイントとしています。特に、CP/M-80の最新バージョンであるCP/M3.0（＝CP/M-Plus）はバンク機能を持ち、64180内蔵MMUと組み合わせることによりプログラムエリアを拡張できるもので、C言語、BASIC言語、FORTRANなどCP/M応用プログラムを使うプログラム開発では必須です（図15・1）。

さらに、CP/Mにはマクロアセンブラ（MACRO-80）などの基本プログラムもあり、このシステムだけで開発装置としても活用できるとともに、実際に64180レベルでの評価を行うことにも利用できます。

システムをインプリメントするにあたっては、64180内蔵のI/O以外の外部I/Oを最小とすることを目標とし、以下の2つのI/Oだけとしています。

- ・フロッピーディスクインタフェース
- ・プリンタインタフェース

フロッピーディスクインタフェースは、CP/Mがディスクオペレーティングシステムであることから必要です。フロッピーディスクのデータ転送速度は250KBPSで、PIOモードの転送でも可能ですが、MPUのスループットを向上させるため64180のDMACと組み合わせて用います（DMA転送）。

プリンタインタフェースはパソコン機能の1つとしてもたせていて、これらのI/Oインタフェースは外部コネクタにより周辺機器と接続します（図15・2）。

15・2 システム構成

〔1〕 I/O の 構 成 システム構成は、15・1 節で述べた I/O 以外にキー入力と表示出力のために 64180 の ASCI があり、ASCI の他方 (ch0) は他のシステムとの通信用として用いています。いずれも RS-232C インタフェース仕様としてドライバ/レシーバ IC を用いています。各部の構成について以下に述べます (図 15・3)。

(1) フロッピーディスクインタフェース：データセパレータ (VFO 回路) を内蔵して、外付部品が少ない HD63265 を用います。データ転送は DMA 転送とします。HD63265 は従来の FDC に対して DMA 転送要求からのデータ転送が速く、64180 を用いた場合に要求をサンプルしてから DMA 転送サイクルを遅らせる必要がなく、外部回路が不要となります。

(2) プリンタインタフェース：セントロニクス 8 ビットパラレルインタフェースに準拠しています。データ転送は専用 LSI ではなく、8 ビットラッチを用いて小形化を行うため、ソフトウェアでサポートします。

(3) RS-232C シリアルインタフェース：64180 の ASCI を使います。ドライバ/レシーバ IC として 75188/189 を用いるため、±12V 電源が必要です。

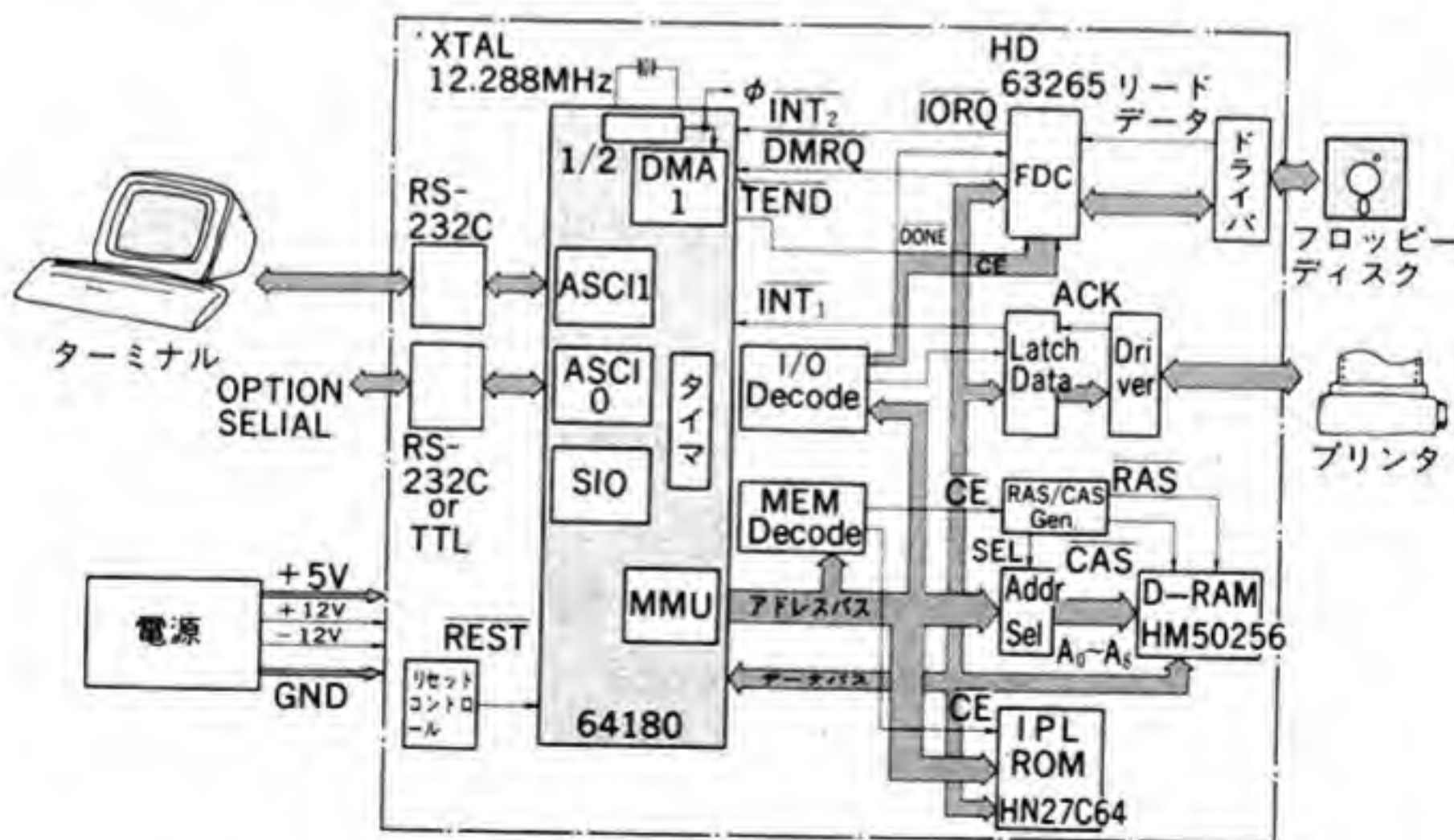


図 15・3 ブロック図

〔2〕 メ モ リ 構 成 CP/Mを走らせる場合、後述する BIOS, BDOS, CCP などのメモリ領域の割当てが重要ですが、64180 内蔵 MMU を用いることによって、物理アドレスに割りつけているメモリを任意の論理アドレスに割り当てることができるため、物理アドレスに対するメモリ構成を意識する必要がありません。

CP/Mを走らせるためには、以下の2つのメモリ構成とします。

(1) DRAM: DRAMとして、HM50256 (256 K×1ビット) を8個用いて256 K バイトのエリアとします。このエリアは主にロードされた CP/M が常駐し、TPA と呼ばれる応用プログラム領域として使うほか、RAM ディスクとして用います。

(2) ROM: 27512 を用います。CP/Mをファイルからロードするためのコールドスタートローダプログラムと簡易モニタが入ります。

15・3 I/O メモリインタフェース

(1) I/O インタフェース：このシステムでは、外部 I/O としてはフロッピーディスクインタフェースとプリンタインタフェースで、I/O サイクルのタイミングは FDC のアクセス時間 (t_{DDR}) に依存します。HD63265 のアクセス時間は 140 ns で、64180 の I/O サイクルは 4 ステート ($(1/6.144\text{ MHz}) \times 4 = 651\text{ ns}$) で、プログラマブルウェイトステートの挿入は不要です。また、13・3 節で述べたように、FDC のデータ転送をデュアルアドレス方式としているため、DMA 転送アドレス (\overline{DACK} 信号が発生) を別の I/O アドレス (A_0 番地) に割りつけます。

プリンタインタフェースは、データ転送信号 (\overline{STROBE} 信号) をソフトウェアで発生させています。手順として、 C_0 番地 (I/O ライト) にデータを書き込んだ後、 C_1 番地に同一データを書き込むことにより \overline{STROBE} 信号を生成させます。つまり、アドレス $A_0=0$ の状態で "H"、 $A_0=1$ の状態で "L" となり、I/O アクセスを 3 回行うことでプリンタにデータを転送させます (図 15・4)。

(2) メモリインタフェース：ROM と DRAM があります。ROM インタフェースの \overline{CE} 信号はアドレスのデコードにより作り、また \overline{OE} 信号は \overline{ME} 信号とします。したがって ROM はアドレス信号 $A_{18}=\text{"L"}$ かつ \overline{ME} 信号 = "L" のと

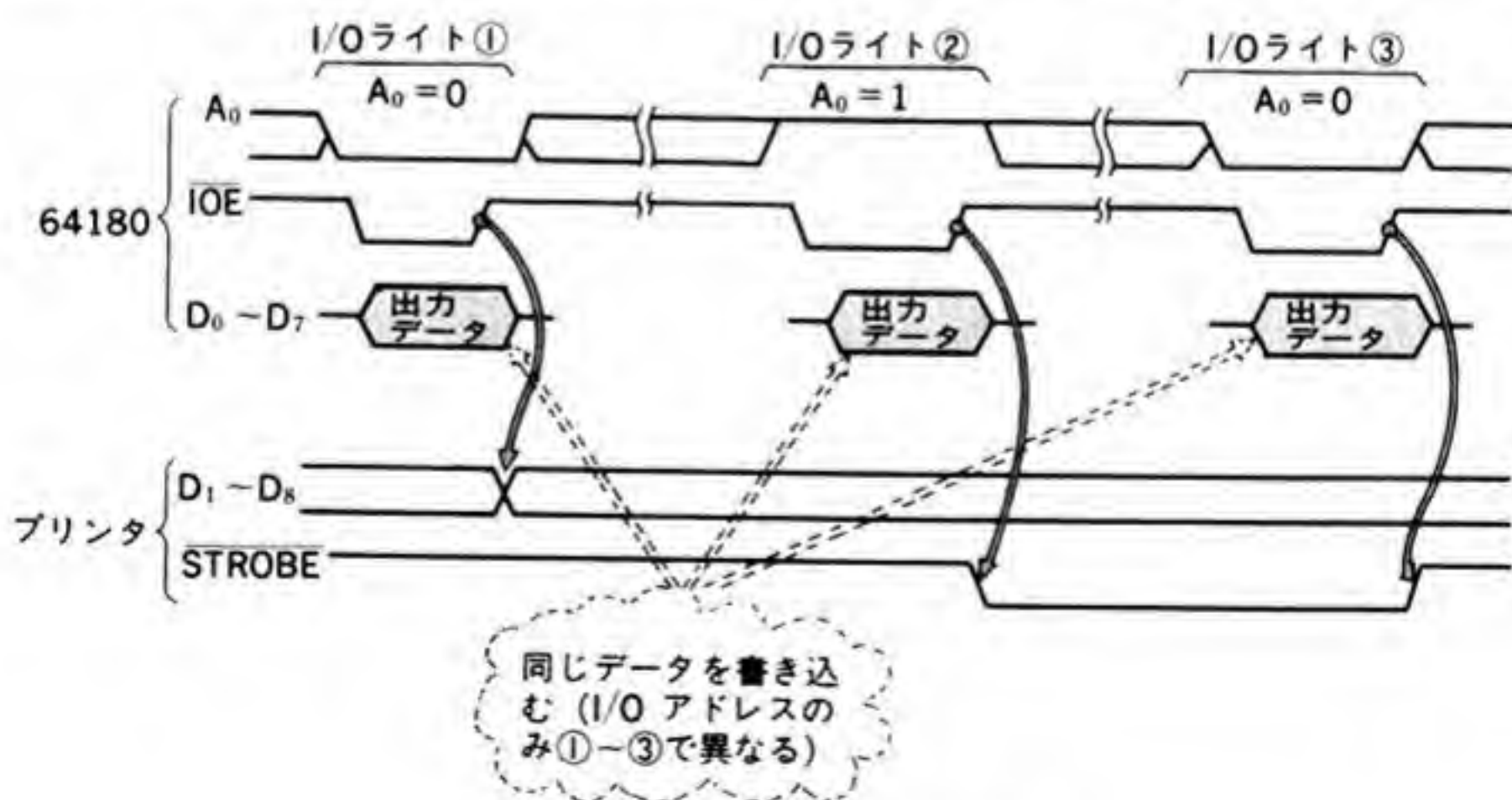


図 15・4 プリンタ出力シーケンス

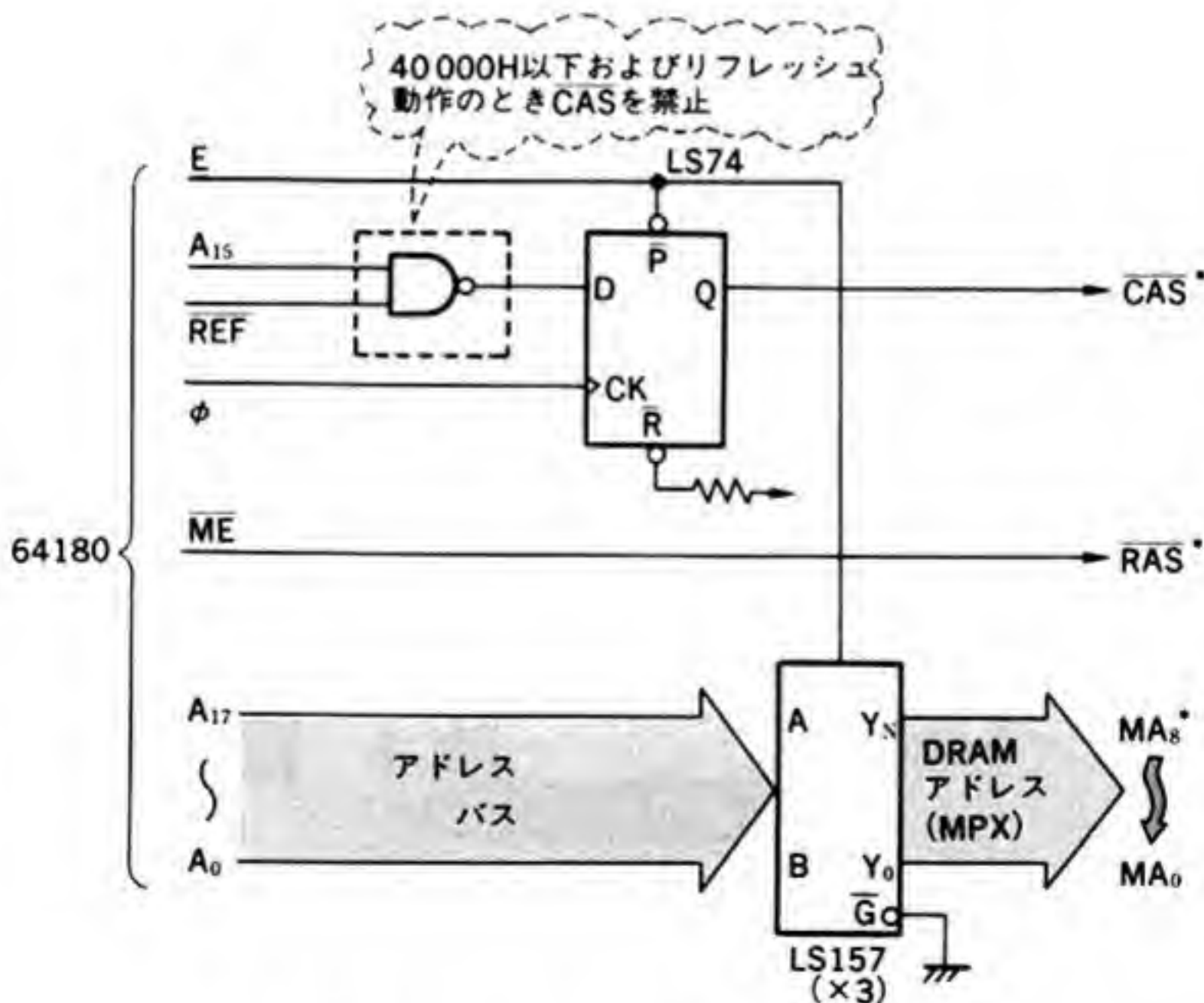
きにアクセスされます。これは回路を省略するために \overline{RD} 信号と論理をとってなく、メモリライトアクセスではデータがぶつかります。

DRAM インタフェースは 13・1 節と同様に、E クロックでロウ/カラムアドレスの切替えタイミングとしているため、 \overline{CAS} 信号を作るのに T_w ステートの挿入が必要となり、メモリアクセスサイクルは、 T_1 、 T_2 、 T_w 、 T_3 の 4 ステートとなります。また、DRAM のリフレッシュは RAS-Only-Refresh モードを用いて、4 ms/256 行アドレス間隔のリフレッシュを行います。リフレッシュ時には、 \overline{RAS} 信号が発生するだけで \overline{CAS} 信号は発生しません。リフレッシュ間隔は下式の条件で設定しています (図 15・5)。

$$\text{リフレッシュ挿入間隔} = N(\text{ステート}) \times \frac{1}{\text{動作周波数}(f)}$$

$$= 80 \times \frac{1}{6.144 \text{ MHz}}$$

$$= 13.02 \mu\text{s} < 15.63 \mu\text{s} \left(= \frac{4 \text{ ms}}{256} \right)$$



注)・DRAM用のダンピング抵抗は省略してあります。

図 15・5 $\overline{RAS}/\overline{CAS}$ 回路例

15・4 CP/M-Plus

CP/Mは80系の8ビットマイコンの標準OSとして多用されており、数多くのアプリケーションソフトウェア財産があります。なかでもCP/M-Plusは、8ビットCPU用OSの中で最高の機能をもっており、その大きな特長として次の2つがあります(図15・6)。

- (1) バンクメモリシステムによるTPA(ユーザプログラムエリア)の拡大
- (2) ディレクトリハッシングとLRUバッファリング技術(ディスクキャッシュ)によるディスクアクセスのスループットの改善

このCP/M-Plusは、汎用OSとして移植性を高めるため、OS全体を次の2つの部分に分けています。

- (1) BDOS (Basic Disk Operating System) : CP/M-Plusの中心部分で、主にディスクに格納されているファイルの管理を行っています。また、論理的な周辺装置に対する文字の入出力も行います。
- (2) BIOS (Basic I/O System) : 文字入出力装置、ディスクドライブ装置などの、周辺装置の制御のようなハードウェアの操作を行います。

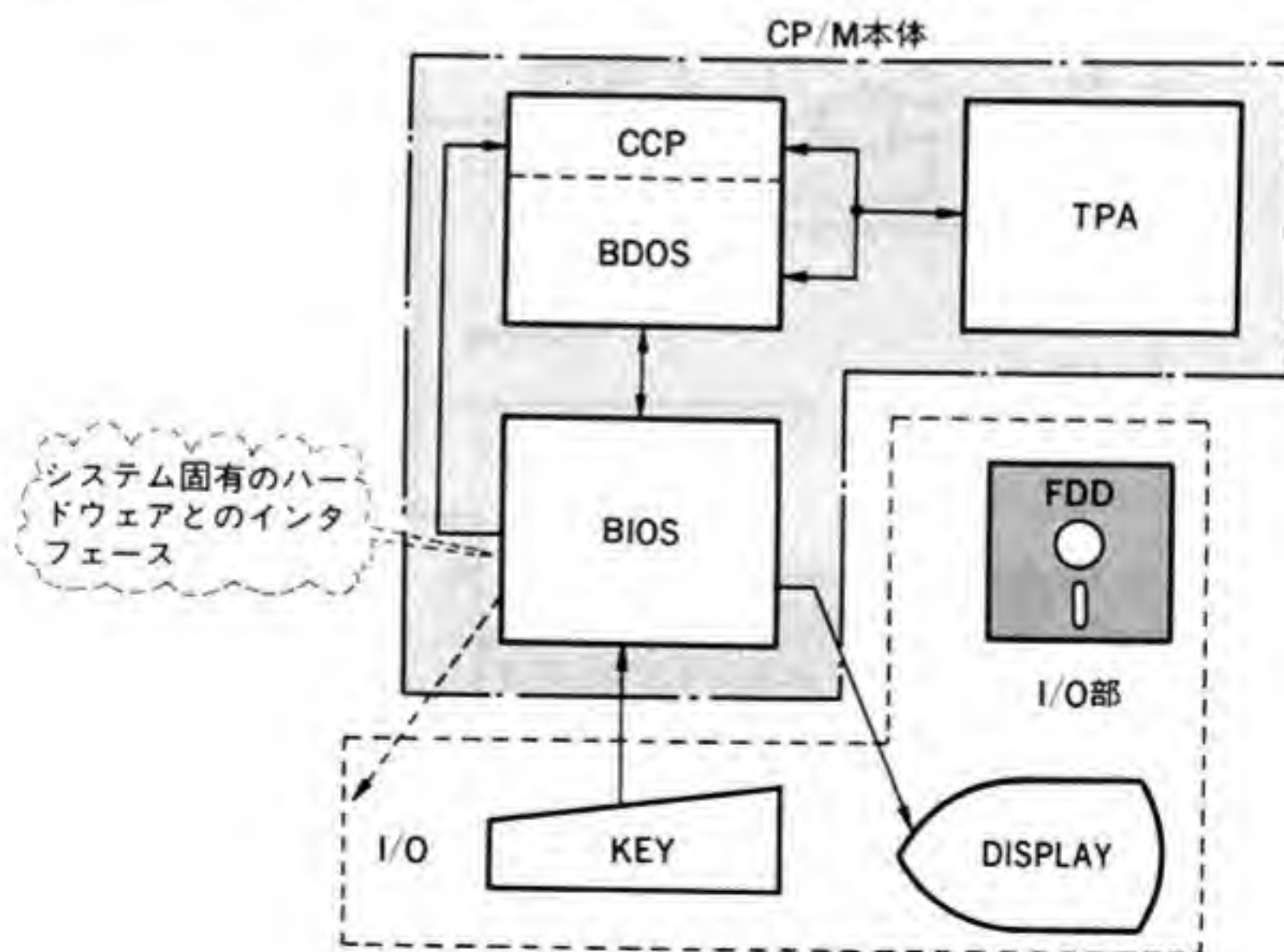


図15・6 CP/Mの構成と処理の流れ

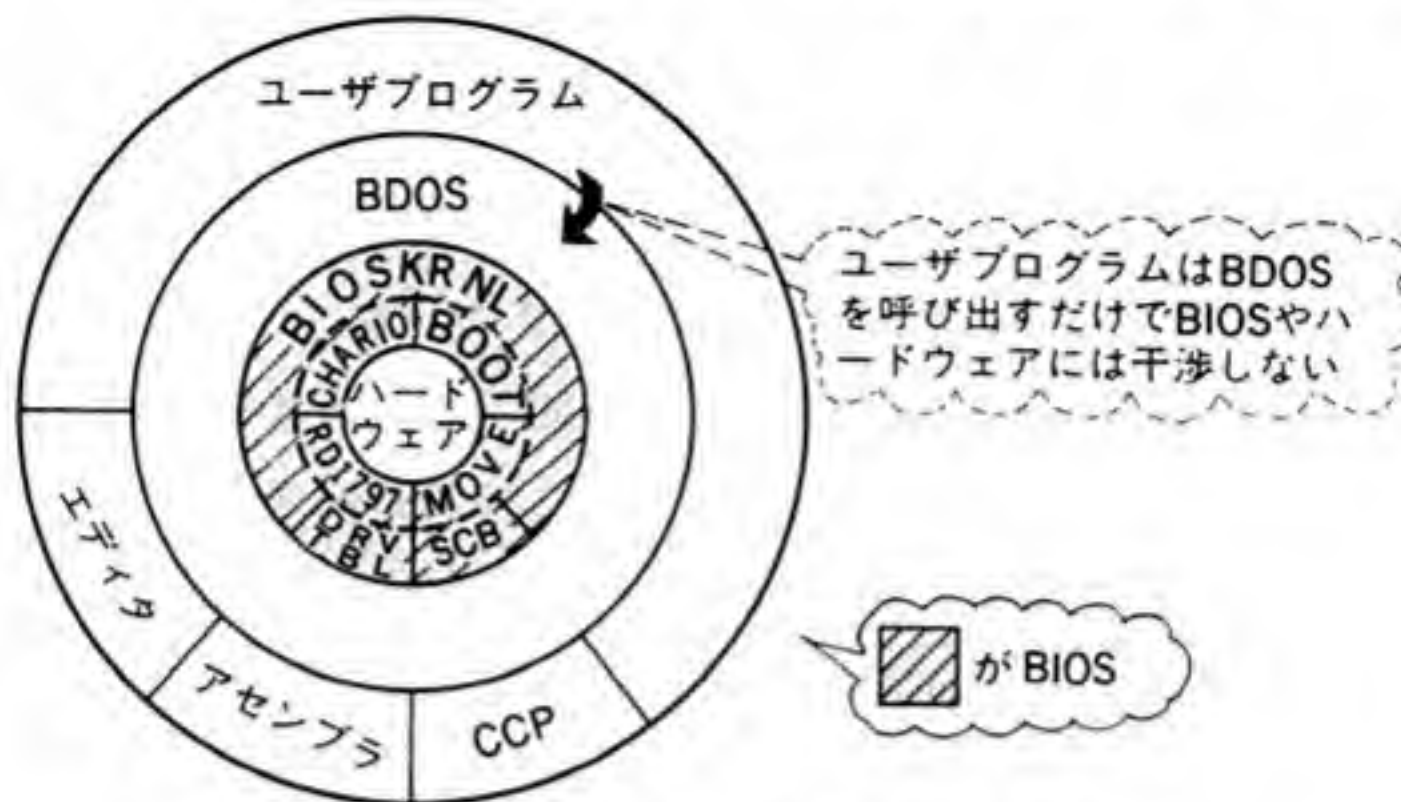


図 15・7 ハードウェア、BIOS、BDOS の関係

以上の2つの部分により、CP/M-Plus システム全体は層状の構造になり、アセンブラやエディタなどのユーティリティプログラム（ユーザプログラム）は BIOS やハードウェアを操作することなく、BDOS の機能を用いるだけで動作します（図 15・7）。

64180 に搭載されている MMU、DMAC は、CP/M-Plus の特長であるバンクメモリ、ディスク、バッファリングを効率的に実現し、CP/M-Plus の能力をさらに引き上げます。しかし、この機能も BIOS の作り方により変わります。64180 の内蔵 I/O を活用し、CP/M-Plus の能力を十分に引き出せるような BIOS を作しましょう。

なお、付録 5 に 180 カードパソコンのハードウェア仕様を示しました。また、本書の見返しに回路構成図を示したので参照して下さい。

15・5 BIOS の 構 成

DRI 社から供給される BIOS のソースプログラムは、主な機能ごとに7つのモジュールに分割されています。この中で、MOVE、CHARIO、DRIVE モジュールをハードウェアにあわせて作成します。これ以外のモジュールは直接ハードウェアに関係することがなく、変更する所も少しだけです。CP/M-Plus の能力を高めるうえで、BIOS の作り方は重要です。64180 の機能を使いこなすことが能力を高める大きな要因です。MOVE、CHARIO、DRIVE の3つのモジュールは、以下に示すようにそれぞれ重要なポイントがあります(表15・1)。

〔1〕 **MOVE** このモジュールにはメモリ↔メモリ間の DMA 転送と、バンク切替えの機能があります。バンクの構成方法は CP/M-Plus のメモリマップの

表 15・1 BIOS のモジュール

モジュール名	機 能
BIOSKRNL	BIOS コールのジャンプテーブルがあり、他のモジュールのルーチンを呼び出す BIOS の中核部分。文字入出力装置やディスクドライブ装置の論理装置—物理装置変換や切替えも行う。移植時にも内容は変更しない。
BOOT	各種文字入出力、ディスクドライブ装置のイニシャライズや CCP コマンドのロード・リロードを行う。この CCP コマンドのリロードは内蔵 DMAC を使うと高速になる。他にタイマルーチンがあるが内蔵タイマの割込みでタイマのカウントアップを行うときには何もしない。
MOVE	バンクの切替え、メモリデータのブロック転送を行う。特に、ブロック転送はシステムを高速にする。
CHARIO	文字入出力装置のイニシャライズ、入出力動作を行う。カードパソコンでは RS-232C が2チャンネルとプリンタ用のチャンネルがある。
DRVTBL	CP/M-Plus の論理ドライブである A-P と物理ドライブの関係を定義する。このテーブル上のドライブがすべての操作 (GENCPM も) の対象となる。
DRIVE (FD1797SD)	ディスクドライブのイニシャライズ、セクタリード/ライトを行う。CP/M-2.2 まではこのルーチン内でセクタのブロック/デブロックを行っていたが、CP/M-Plus ではこの作業を BDOS が行っている。
SCB	BDOS と BIOS の共通の変数エリア。主に BIOS の重要な設定値が入っている。

考え方や DMA 転送時の問題もあるので、よく考えて決めます。特に、メモリ↔メモリ間の DMA 転送はメモリの効率やスピードを決める重要な要素です。

〔2〕 **CHARIO** このモジュールは 1 文字単位の入出力を行います。カードパソコンでは 64180 内蔵の ASCII が 2 つとプリンタインタフェースを制御します。ASCII はフラグの状態を見るだけで送受信できますが、プリンタは割込みを使います。このため、CP/M-Plus システムを高度にするためには、ASCII の割込み機能を使ってリングバッファができるようにします。

〔3〕 **DRIVE** このモジュールでは FDC の制御を行います。ここでも FDC からの割込みを使って、コマンドの終了以外にフロッピーディスクドライブの READY 状態のチェックを行います。

以上の BIOS を作り、アセンブル・リンクした後、DRI 社から提供される GENCPM を動作させると、CPM3.SYS が作られます。このように CP/M-Plus の移植には、少なくとも CP/M-2.X 以上のマシンを使います。

15・6 MMU と CP/M-Plus のバンク

CP/M-Plus のメモリ空間は、64 K バイトの論理アドレス空間を 2 つに分け、アドレスの FFFFH 側の約 4 K バイトを、コモンエリア、0 側の約 60 K バイトをバンクエリアとして配置しています。バンクエリアは 2~16 まで置くことができ、ソフトウェアにより指定のバンクエリアと切り替えます。コモンエリアはバンクエリアの切替えに関係なく、各バンクに共通となります。バンク 0 とバンク 1 には特別の役割があり、その他のバンクは、ディスクバッファや RAM ディスクとして使用できます。バンク 0 は CP/M-Plus の本体部が常駐し、バンク 1 はユーザのプログラムが置かれる TPA (トランジェント・プログラム・エリア) です (図 15・8、図 15・9)。

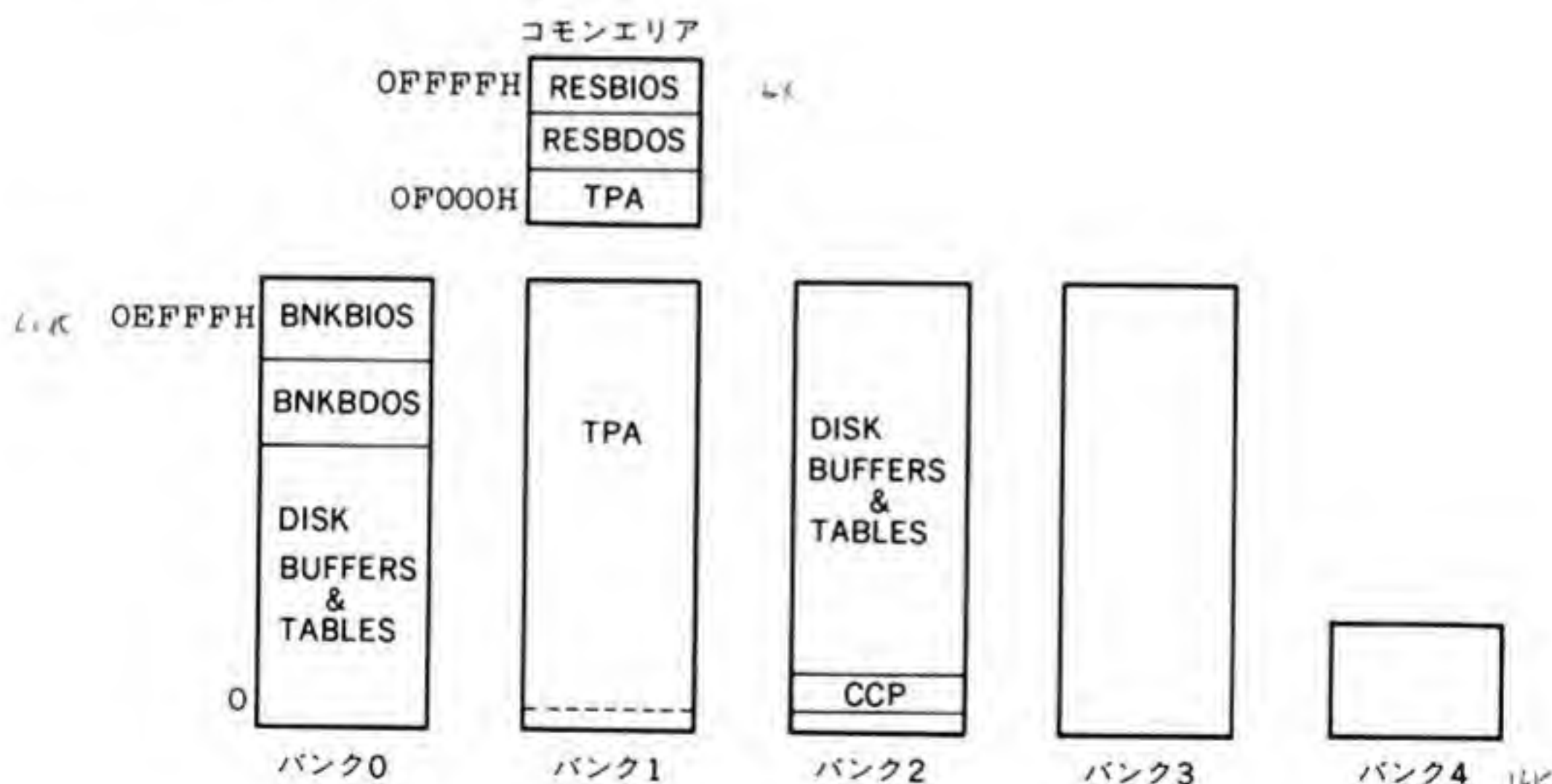


図 15・8 CP/M-Plus のバンク方式

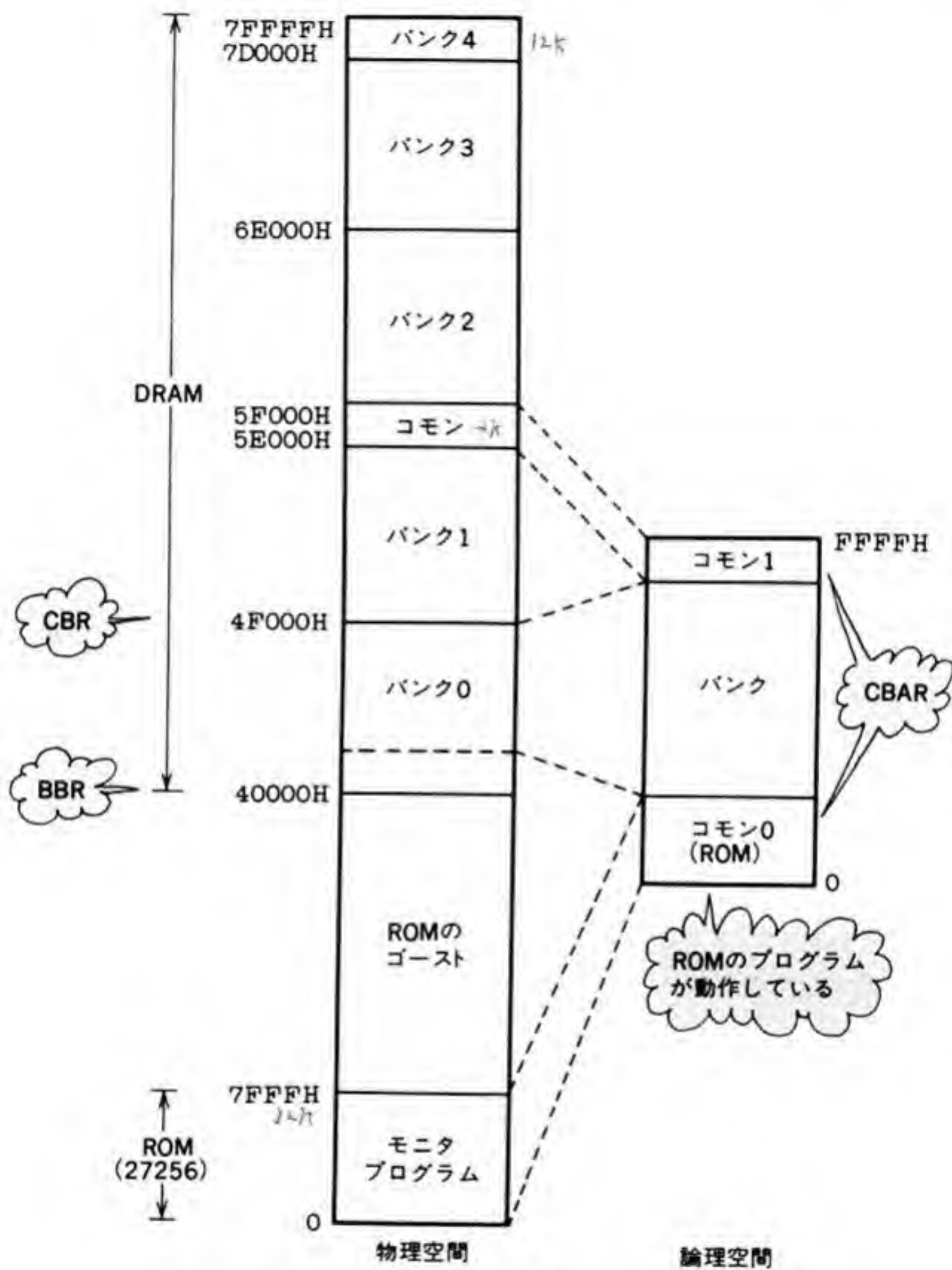


図 15・9 物理メモリマップと論理バンクの対応

15. 180 カードパソコン

一方、64180 は、64 K バイトの論理アドレスを MMU により 1 M バイトの物理アドレスに拡張しています。1 M バイトの空間は、連続的で 64 K バイトごとに区切られていません。この 1 M バイトの空間を適切なところで分割し、CP/M-Plus に対応したバンクメモリにします。これは、64180 の MMU を用いれば簡単に分割できますが、64180 の物理メモリ空間と CP/M-Plus の論理メモリ空間をどのように対応させるかは、プログラムの実行効率のうえで重要です。その構成によっては次節の問題がおこることもあります。

カードパソコンでは、64180 の MMU により 64 K バイトのメモリ空間を、4 K バイトのコモンエリアと 60 K バイトのバンクエリアに分割しています。また、カードパソコンは HM50256 を 8 個用いており、256 K バイトの物理メモリ空間があります。これをコモンエリア 1 つとバンクエリア 5 つに分割します。バンクエリアの 1 つは 12 K バイトとなります。

分割は BIOS ソフトウェア上での定義だけで物理空間と論理空間の対応が決まるので、ハードウェアでの対応が必要なわけではありません。

15・7 DMA 転送とバンク

CP/M-Plus はディスク・バッファによりディスク・アクセスのスループットを向上させていますが、この機能ではメモリ-FDC 間の DMA 転送およびメモリ↔メモリ間の DMA 転送が多用されます、このように DMA 転送は便利な機能ですが、前節の論理・物理のメモリ空間の対応でも考慮すべき点があります。

前節で述べたように、CP/M-Plus は 64180 の物理メモリ空間を分割したバンク方式による論理メモリ空間です。CP/M-Plus の本体である BDOS や TPA 内のユーザプログラムは、この論理的なメモリ空間の上で動作します。しかし、DMAC は MMU を通して物理アドレスを作っていません。直接、1 M バイトの物理アドレス空間に対して DMA 転送を実行します (図 15・10)。

前節のカードパソコンでのメモリ分割例で、コモンエリアとバンク 0 が物理空間上で連続した配置とした場合、CP/M-Plus はバンク 1 に TPA を置きますが、コモンエリアにも TPA の続きが置かれます。この TPA 上でアプリケーションのプログラムが動作しているとき、プログラムは TPA の端までディスクからデータをロードします。これは DMA 転送により実行されますが、データが多ければ E000H から F1FFFH まで一度に転送します。このときの DMAC へ設定する物理アドレスは、バンク 1 に対応した値です。転送の最初の 4 K バイト (E000H ~ EFFFH) は正常に行われますが、F000H から F1FFFH へ転送するはずのデータはコモンエリアではなく、バンク 2 に転送されバンク 2 の内容を破壊します。DMAC は、プログラム上で定義しているメモリの分割を考慮して転送しません。物理空間に対して転送します。

カードパソコンでは以上の問題により前項のメモリ配置とし、バンクエリア 1 とコモンエリアを論理空間だけでなく物理空間上でも連続した配置にしました。

15. 180 カードパソコン

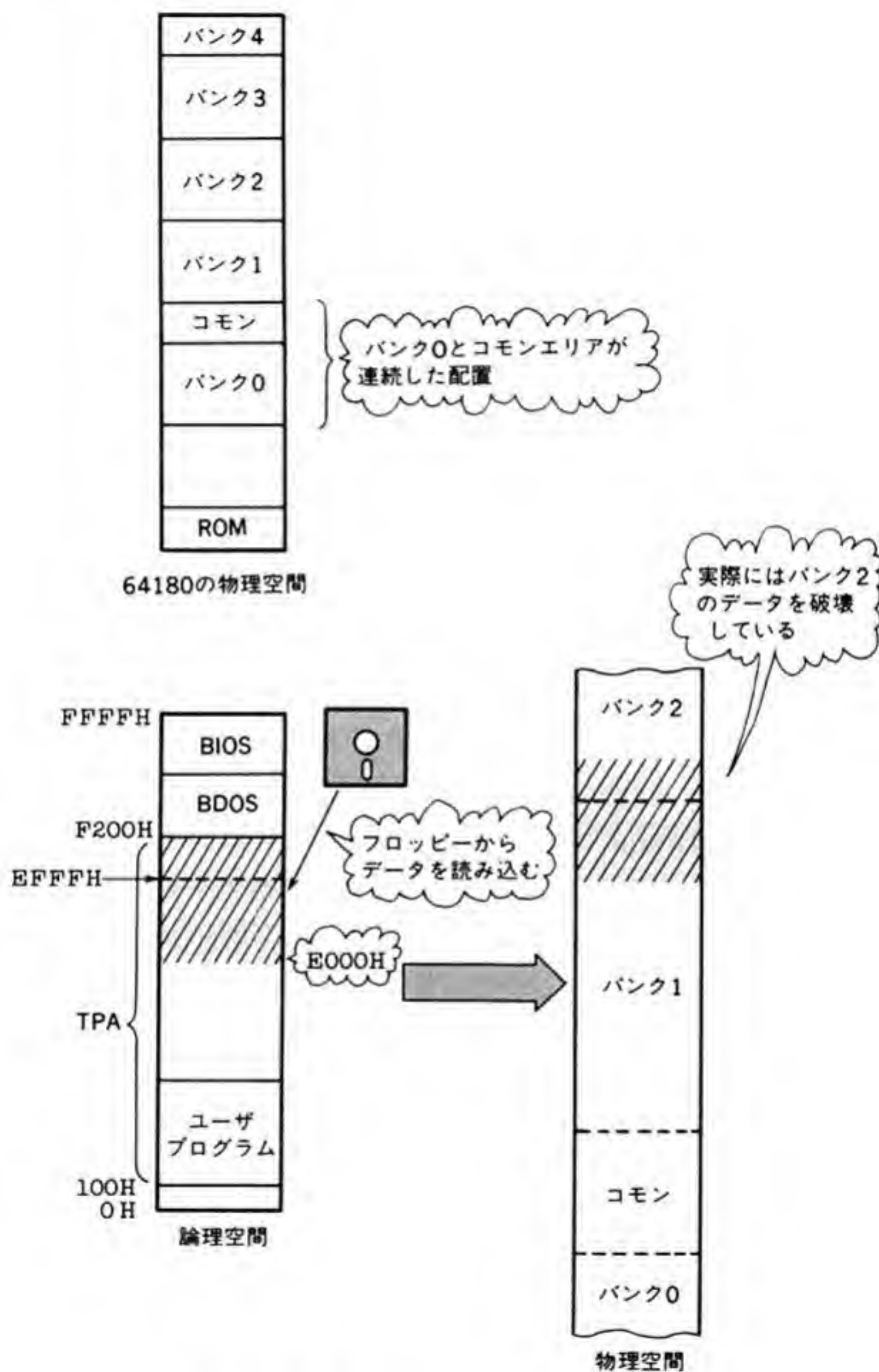


図 15・10 DMA 転送の問題がおこる例

15・8 CP/M-Plus 起動時の メモリマップの変化

CP/M-Plus のバンクメモリの具体的な動作例として、CP/M-Plus 起動時のメモリマップの変化を説明します。CP/M-Plus は DRAM 上にロードされ動作しますが、64180 のリセットから CP/M-Plus が起動するまでの動作は少々複雑です。

① 64180 がリセットされると、MMU などが初期化され、64180 からは ROM だけがアクセスできます。ROM には IPL (イニシャルプログラムローダ) が書き込まれています。

② ROM のプログラムが動作し、MMU のレジスタを設定します。

③ フロッピーディスクの 0 トラック 1 セクタからコールドスタートローダを読み込み、コールドスタートローダにジャンプします。

④ コールドスタートローダは、最初に MMU の CBAR を設定し直し ROM を論理空間からはずします。

⑤ 次に、フロッピーディスクのトラック 0 セクタ 2 ～セクタ 16 に入っている CP/M ロードを読み込み、CP/M ロードにジャンプします。

⑥ CP/M ロードはフロッピーディスクの CPM3.SYS ファイルを捜し、ファイルのヘッダレコード中の配置情報により、BNKBDOS、BNKBIOS、RESBDOS、RESBIOS をそれぞれ配置しながら読み込みます。その後、BIOS の BOOT コールにジャンプします。

⑦ BIOS の BOOT コールは周辺 I/O のイニシャライズ後、バンク 1 に切り替えます。

以上で CP/M-Plus の動作準備が完了します。この後、CCP コマンドを実行し "A>" のプロンプトが表示され、コマンド入力待ちになります (図 15・11)。

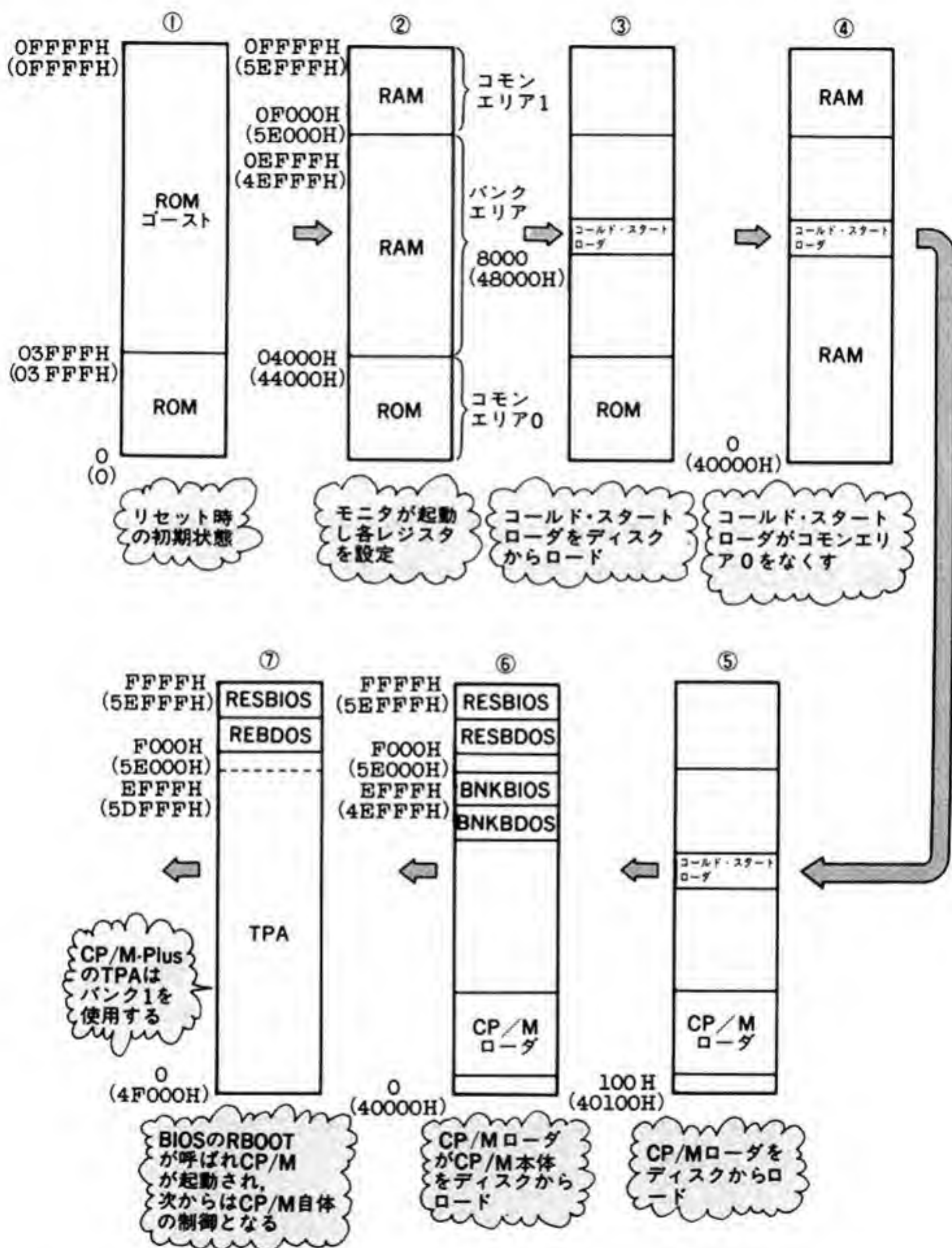


図 15・11 CP/M-Plus 起動時の論理メモリ内容の変化—()は物理アドレス—

15・9 64180 内蔵 I/O と BIOS

BIOS の中で 64180 の内蔵 I/O はどのようにプログラミングされているか、MMU と DMAC、ASCII について取り上げます。

〔1〕 **MMU** バンク切替えの BIOS コール、SETBNK で設定します。各バンクに対応した BBR の値はまとめてテーブルにしておき、バンク番号から BBR 値を変換できるようにします。このテーブルは DMAC を設定するルーチンでも用います。CP/M-Plus システムのメモリ構成を決める重要なルーチンですが、内容は図のとおり簡単です。

〔2〕 **DMAC** DMAC はフロッピーディスクの BIOS コール、READ/WRITE やメモリ↔メモリ間のデータ転送の BIOS コール・MOVE で用います。ここでは DMAC の設定時に共通な論理→物理アドレス変換の部分だけを取り上げます。

BIOS がコールされたときに指定された転送アドレスはバンク番号と論理アドレスです。しかし、DMAC は物理アドレスで転送するので、バンク番号と論理アドレスを計算して物理アドレスにします。この計算は MMU の設定で用いたテーブルからバンク番号に対応した値を取り出し、アドレスの上位と加算します。計算結果は物理アドレスとなりますが、DMA 転送はコモンエリアに対して行われることもあるので、このときにはバンク番号を 1 にした計算をします。

ソースアドレス、ディスティネーションアドレスともアドレスを設定した後、DMAC をスタートさせます。メモリ↔メモリ間転送の場合、転送はバーストモードで行われ、プログラムで転送終了を待つ必要はありません。

〔3〕 **ASCII** ASCII は文字入出力を行う BIOS コール、CONIN、CONOUT などで用います。プログラム例は ASCII1 からの入力ルーチンです。プログラムは、文字が入力されたかをチェックするルーチンと、入力された文字をレジスタから取り出すルーチンに分かれます。このように入出力プログラムでも、内蔵 I/O では短いプログラムで実現できます (図 15・12)。

15. 180 カードパソコン

① バンク切替えプログラム

```
?bank:  PUSH DE
        PUSH HL
        LD  D,0
        LD  E,A
        LD  HL,bank table
        ADD HL,DE
        LD  A,(HL)
        OUT (bbr),A
        POP HL
        POP DE
        RET
```

バンク番号に対応したテーブル値をロード
←BBR にセットしバンク切替え

banktable: DB 40H, 4FH, 5FH, 6EH, 7DH

② DMA 論理-物理アドレス変換プログラム

```
LD  HL,(dmaaddress)
PUSH HL
LD  A,(bank)
LD  C,A
LD  A,0FFH
CP  H
JR  Z,cmn
JR  NC,n_cmn
cmn:  LD  C,1
n_cmn: LD B,0
        LD  HL,banktable
        ADD HL,BC
        LD  C,(HL)
        LD  B,16
        MLT BC
        POP HL
        LD  A,H
        ADD A,C
        LD  H,A
        LD  A,B
        ADC A,0
```

DMA アドレスがコモンエリア内かをチェック
←コモンエリア内であればバンク番号を1にする
バンク番号に対応したテーブル値をロード
テーブル値×16
テーブル値と DMA アドレスを加算

③ ASCII 入力フラグテストプログラム

```
ascii_st: IN0  A,(stat1)
        AND  80H
        RET  Z
        OR  0FFH
        RET
```

←RDRF フラグをテスト
←フラグが0ならAを0にしてリターン
←フラグが1ならAをFFHにしてリターン

④ ASCII 入力プログラム

```
ascii_in: CALL ascii_st
        JR  Z,ascii_in
        IN0 A,(tsr1)
        RET
```

文字を受信するまで待つ
←文字をロード

図 15・12 内蔵I/Oのプログラム

付 録

1 命 令 一 覧

命令セットの一覧表の中で使用される記号を以下に説明します。

1. レジスタ指定

g, g', ww, xx, yy, zz はレジスタを指定する記号です。 g, g' は8ビットのレジスタ、 ww, xx, yy, zz は16ビットのレジスタペアを指定します。おのこの対応するレジスタは下記のとおりです。

g, g'	Reg.	ww	Reg.	xx	Reg.	yy	Reg.	zz	Reg.
000	B	00	BC	00	BC	00	BC	00	BC
001	C	01	DE	01	DE	01	DE	01	DE
010	D	10	HL	10	IX	10	IY	10	HL
011	E	11	SP	11	SP	11	SP	11	AF
100	H								
101	L								
111	A								

注) ww, xx, yy, zz にHまたはLを付加することにより、16ビットレジスタの上位8ビットまたは下位8ビットを表します(例、 wwH, IXL)。

2. ビット指定

b は、ビット操作命令におけるビットオペランドが何ビット目かを指定する記号です。おのこの対応するビットは右表のとおりです。

b	Bit
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

3. コンディション指定

f は、演算の結果を判定して命令を実行する場合の条件を指定する記号です。おのこの対応するコンディションは右表のとおりです。

f	Condition
000	NZ non zero
001	Z zero
010	NC non carry
011	C carry
100	PO parity odd
101	PE parity even
110	P sign plus
111	M sign minus

4. リスタートアドレス

v は、リスタート命令のリスタートアドレスを指定する記号です。おのこの対応するアドレスは右表のとおりです。

v	Address
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

5. フラグ

フラグの変化を示す記号について説明します。

- ・：その命令によってフラグは変化しない
- ×：その命令によってフラグの変化は不定
- ↑↓：その命令によって演算結果に従いフラグは変化
- S：その命令によって“1”にセット
- R：その命令によって“0”にリセット
- P：その命令によってパリティフラグとして変化
- V：その命令によってオーバフローフラグとして変化

6. その他

()_M：() 内の内容をアドレスとするメモリを表します。

()_I : () 内の内容をアドレスとする I/O を表します

m or n : 8 ビットの値

m n : 16 ビットの値

r : r の添字がついていると、8 ビットレジスタを表します

R : R の添字がついていると、16 ビットレジスタを表します

b · ()_M : () 内の内容をアドレスとするメモリの b で指定されるビットを表します

b : gr : gr で指定されるレジスタの内容の b で指定されるビットを表します

d or j : 符号つき 8 ビットディスプレースメント

S : ソースアドレッシングモード

D : ディスティネーションアドレッシングモード

· : AND 演算

⊕ : OR 演算

⊕ : EXCLUSIVE OR 演算

(1) 演算命令

1. 算術論理演算 (8-bit)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
ADD	ADD A, g	10 000 g	1	4	Ar+gr→Ar	↑	↑	↑	V	R	↑
	ADD A, (HL)	10 000 110	1	6	Ar+(HL) _M →Ar	↑	↑	↑	V	R	↑
	ADD A, m	11 000 110 <m>	2	6	Ar+m→Ar	↑	↑	↑	V	R	↑
	ADD A, (IX+d)	11 011 101 10 000 110 <d>	3	14	Ar+(IX+d) _M →Ar	↑	↑	↑	V	R	↑
	ADD A, (IY+d)	11 111 101 10 000 110 <d>	3	14	Ar+(IY+d) _M →Ar	↑	↑	↑	V	R	↑
ADC	ADC A, g	10 001 g	1	4	Ar+gr+c→Ar	↑	↑	↑	V	R	↑
	ADC A, (HL)	10 001 110	1	6	Ar+(HL) _M +c→Ar	↑	↑	↑	V	R	↑
	ADC A, m	11 001 110 <m>	2	6	Ar+m+c→Ar	↑	↑	↑	V	R	↑
	ADC A, (IX+d)	11 011 101 10 001 110 <d>	3	14	Ar+(IX+d) _M +c→Ar	↑	↑	↑	V	R	↑
	ADC A, (IY+d)	11 111 101 10 001 110 <d>	3	14	Ar+(IY+d) _M +c→Ar	↑	↑	↑	V	R	↑

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
AND	AND g	10 100 g	1	4	$Ar \cdot gr \rightarrow Ar$	↑	↑	S	P	R	R
	AND (HL)	10 100 110	1	6	$Ar \cdot (HL)_M \rightarrow Ar$	↑	↑	S	P	R	R
	AND m	11 100 110 (m)	2	6	$Ar \cdot m \rightarrow Ar$	↑	↑	S	P	R	R
	AND (IX+d)	11 011 101 10 100 110 (d)	3	14	$Ar \cdot (IX+d)_M \rightarrow Ar$	↑	↑	S	P	R	R
	AND (IY+d)	11 111 101 10 100 110 (d)	3	14	$Ar \cdot (IY+d)_M \rightarrow Ar$	↑	↑	S	P	R	R
Compare	CP g	10 111 g	1	4	$Ar - gr$	↑	↑	↑	V	S	↑
	CP (HL)	10 111 110	1	6	$Ar - (HL)_M$	↑	↑	↑	V	S	↑
	CP m	11 111 110 (m)	2	6	$Ar - m$	↑	↑	↑	V	S	↑
	CP (IX+d)	11 011 101 10 111 110 (d)	3	14	$Ar - (IX+d)_M$	↑	↑	↑	V	S	↑
	CP (IY+d)	11 111 101 10 111 110 (d)	3	14	$Ar - (IY+d)_M$	↑	↑	↑	V	S	↑
COMPLEMENT	CPL	00 101 111	1	3	$\overline{Ar} \rightarrow Ar$.	.	S	.	S	.
DEC	DEC g	00 g 101	1	4	$gr - 1 \rightarrow gr$	↑	↑	↑	V	S	.
	DEC (HL)	00 110 101	1	10	$(HL)_M - 1 \rightarrow (HL)_M$	↑	↑	↑	V	S	.
	DEC (IX+d)	11 011 101 00 110 101 (d)	3	18	$(IX+d)_M - 1 \rightarrow (IX+d)_M$	↑	↑	↑	V	S	.
	DEC (IY+d)	11 111 101 00 110 101 (d)	3	18	$(IY+d)_M - 1 \rightarrow (IY+d)_M$	↑	↑	↑	V	S	.
INC	INC g	00 g 100	1	4	$gr + 1 \rightarrow gr$	↑	↑	↑	V	R	.
	INC (HL)	00 110 100	1	10	$(HL)_M + 1 \rightarrow (HL)_M$	↑	↑	↑	V	R	.
	INC (IX+d)	11 011 101 00 110 100 (d)	3	18	$(IX+d)_M + 1 \rightarrow (IX+d)_M$	↑	↑	↑	V	R	.
	INC (IY+d)	11 111 101 00 110 100 (d)	3	18	$(IY+d)_M + 1 \rightarrow (IY+d)_M$	↑	↑	↑	V	R	.
MULT	MLT _{ww}	11 101 101 01 _{ww} 1 100	2	17	$_{ww}Hr \times _{ww}Lr \rightarrow _{ww}R$
NEGATE	NEG	11 101 101 01 000 100	2	6	$0 - Ar \rightarrow Ar$	↑	↑	↑	V	S	↑

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
OR	OR g	10 110 g	1	4	Ar+gr→Ar	↑	↑	R	P	R	R
	OR (HL)	10 110 110	1	6	Ar+(HL) _M →Ar	↑	↑	R	P	R	R
	OR m	11 110 110 <m>	2	6	Ar+m→Ar	↑	↑	R	P	R	R
	OR (IX+d)	11 011 101 10 110 110 <d>	3	14	Ar+(IX+d) _M →Ar	↑	↑	R	P	R	R
	OR (IY+d)	11 111 101 10 110 110 <d>	3	14	Ar+(IY+d) _M →Ar	↑	↑	R	P	R	R
SUB	SUB g	10 010 g	1	4	Ar-gr→Ar	↑	↑	↑	V	S	↑
	SUB (HL)	10 010 110	1	6	Ar-(HL) _M →Ar	↑	↑	↑	V	S	↑
	SUB m	11 010 110 <m>	2	6	Ar-m→Ar	↑	↑	↑	V	S	↑
	SUB (IX+d)	11 011 101 10 010 110 <d>	3	14	Ar-(IX+d) _M →Ar	↑	↑	↑	V	S	↑
	SUB (IY+d)	11 111 101 10 010 110 <d>	3	14	Ar-(IY+d) _M →Ar	↑	↑	↑	V	S	↑
SUBC	SBC A, g	10 011 g	1	4	Ar-gr-c→Ar	↑	↑	↑	V	S	↑
	SBC A, (HL)	10 011 110	1	6	Ar-(HL) _M -c→Ar	↑	↑	↑	V	S	↑
	SBC A, m	11 011 110 <m>	2	6	Ar-m-c→Ar	↑	↑	↑	V	S	↑
	SBC A, (IX+d)	11 011 101 10 011 110 <d>	3	14	Ar-(IX+d) _M -c→Ar	↑	↑	↑	V	S	↑
	SBC A, (IY+d)	11 111 101 10 011 110 <d>	3	14	Ar-(IY+d) _M -c→Ar	↑	↑	↑	V	S	↑
TEST	TST g	11 101 101 00 g 100	2	7	Ar·gr	↑	↑	S	P	R	R
	TST (HL)	11 101 101 00 110 100	2	10	Ar·(HL) _M	↑	↑	S	P	R	R
	TST m	11 101 101 01 100 100 <m>	3	9	Ar·m	↑	↑	S	P	R	R
XOR	XOR g	10 101 g	1	4	Ar⊕gr→Ar	↑	↑	R	P	R	R
	XOR (HL)	10 101 110	1	6	Ar⊕(HL) _M →Ar	↑	↑	R	P	R	R
	XOR m	11 101 110 <m>	2	6	Ar⊕m→Ar	↑	↑	R	P	R	R
	XOR (IX+d)	11 011 101 10 101 110	3	14	Ar⊕(IX+d) _M →Ar	↑	↑	R	P	R	R

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
	XOR (IY+d)	<d> 11 111 101 10 101 110 <d>	3	14	$Ar \oplus (IY+d)_M \rightarrow Ar$	↓	↓	R	P	R	R

2. ロータートおよびシフト命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ						
						7	6	4	2	1	0	
						S	Z	H	P/V	N	C	
Rotate and Shift Data	RLA	00 010 111	1	3		.	.	R	.	R	↓	
	RL g	11 001 011	2	7		↓	↓	R	P	R	↓	
	RL (HL)	00 010 g	2	13		↓	↓	R	P	R	↓	
		11 001 011				↓	↓	R	P	R	↓	
	RL (IX+d)	00 010 110	4	19		↓	↓	R	P	R	↓	
		11 011 101				↓	↓	R	P	R	↓	
	RL (IY+d)	11 001 011 <d>	4	19		↓	↓	R	P	R	↓	
		00 010 110 11 111 101 11 001 011 <d>				↓	↓	R	P	R	↓	
	RLCA	00 010 110		.	.	R	.	R	↓			
	RLC g	00 000 111		1	3	↓	↓	R	P	R	↓	
	RLC (HL)	11 001 011		2	7	↓	↓	R	P	R	↓	
		00 000 g		2	13	↓	↓	R	P	R	↓	
	RLC (IX+d)	11 001 011		2	13	↓	↓	R	P	R	↓	
		00 000 110	4	19	↓	↓	R	P	R	↓		
	RLC (IY+d)	11 011 101	4	19	↓	↓	R	P	R	↓		
		11 001 011 <d>	4	19		↓	↓	R	P	R	↓	
	RLD	00 000 110	2	16		↓	↓	R	P	R	.	
	RRA	11 101 101	2	16		↓	↓	R	P	R	.	
	RR g	01 101 111	1	3		.	.	R	.	R	↓	
		00 011 111	2	7		↓	↓	R	P	R	↓	
	RR (HL)	11 001 011	2	13		↓	↓	R	P	R	↓	
		00 011 g	2	13		↓	↓	R	P	R	↓	
		11 001 011	2	13	↓	↓	R	P	R	↓		
	00 011 110				↓	↓	R	P	R	↓		

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Rotate and Shift Data	RR (IX+d)	11 011 101 11 001 011 <d> 00 011 110	4	19		↓	↓	R	P	R	↓
	RR (IY+d)	11 111 101 11 001 011 <d> 00 011 110	4	19		↓	↓	R	P	R	↓
	RRCA	00 001 111	1	3		.	.	R	.	R	↓
	RRC g	11 001 011 00 001 g	2	7		↓	↓	R	P	R	↓
	RRC (HL)	11 001 011 00 001 110	2	13		↓	↓	R	P	R	↓
	RRC (IX+d)	11 011 101 11 001 011 <d> 00 001 110	4	19		↓	↓	R	P	R	↓
	RRC (IY+d)	11 111 101 11 001 011 <d> 00 001 110	4	19		↓	↓	R	P	R	↓
	RRD	11 101 101 01 100 111	2	16		↓	↓	R	P	R	.
	SLA g	11 001 011 00 100 g	2	7		↓	↓	R	P	R	↓
	SLA (HL)	11 001 011 00 100 110	2	13		↓	↓	R	P	R	↓
	SLA (IX+d)	11 011 101 11 001 011 <d> 00 100 110	4	19		↓	↓	R	P	R	↓
	SLA (IY+d)	11 111 101 11 001 011 <d> 00 100 110	4	19		↓	↓	R	P	R	↓
	SRA g	11 001 011 00 101 g	2	7		↓	↓	R	P	R	↓
	SRA (HL)	11 001 011 00 101 110	2	13		↓	↓	R	P	R	↓
	SRA (IX+d)	11 011 101 11 001 011 <d> 00 101 110	4	19		↓	↓	R	P	R	↓
	SRA (IY+d)	11 111 101 11 001 011	4	19		↓	↓	R	P	R	↓

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
	SRL g	<d> 00 101 110 11 001 011	2	7		↓	↓	R	P	R	↓
	SRL (HL)	00 111 g 11 001 011 00 111 110	2	3		↓	↓	R	P	R	↓
	SRL (IX+d)	11 011 101 11 001 011 <d>	4	19		↓	↓	R	P	R	↓
	SRL (IY+d)	00 111 110 11 111 101 11 001 011 <d>	4	19		↓	↓	R	P	R	↓
		00 111 110									

3. ビット操作命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Bit Set	SET b.g	11 001 011 11 b g	2	7	1→b.gr
	SET b.(HL)	11 001 011 11 b 110	2	13	1→b.(HL) _M
	SET b.(IX+d)	11 011 101 11 001 011 <d>	4	19	1→b.(IX+d) _M
	SET b.(IY+d)	11 b 110 11 111 101 11 001 011 <d>	4	19	1→b.(IY+d) _M
Bit Reset	RES b.g	11 001 011 10 b g	2	7	0→b.gr
	RES b.(HL)	11 001 011 10 b 110	2	13	0→b.(HL) _M
	RES b.(IX+d)	11 011 101 11 001 011 <d>	4	19	0→b.(IX+d) _M
	RES b.(IY+d)	10 b 110 11 111 101	4	19	0→b.(IY+d) _M

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
		11 001 011 (d) 10 b 110									
Bit Test	BIT b, g	11 001 011 01 b g	2	6	$b \cdot gr \rightarrow z$	X	↓	S	X	R	·
	BIT b, (HL)	11 001 011 01 b 110	2	9	$b \cdot (HL)_R \rightarrow z$	X	↓	S	X	R	·
	BIT b, (IX+d)	11 011 101 11 001 011 (d) 01 b 110	4	15	$b \cdot (IX+d)_R \rightarrow z$	X	↓	S	X	R	·
	BIT b, (IY+d)	11 111 101 11 001 011 (d) 01 b 110	4	15	$b \cdot (IY+d)_R \rightarrow z$	X	↓	S	X	R	·

4. 演 算 命 令 (16-bit)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
ADD	ADD HL, WW	00 WW 1 001	1	7	$HL_R + WW_R \rightarrow HL_R$	·	·	X	·	R	↓
	ADD IX, XX	11 011 101	2	10	$IX_R + XX_R \rightarrow IX_R$	·	·	X	·	R	↓
	ADD IY, YY	00 XX 1 001	2	10	$IY_R + YY_R \rightarrow IY_R$	·	·	X	·	R	↓
		11 111 101 00 YY 1 001	2	10							
ADC	ADC HL, WW	11 101 101 01 WW 1 010	2	10	$HL_R + WW_R + C \rightarrow HL_R$	↓	↓	X	V	R	↓
DEC	DEC WW	00 WW 1 011	1	4	$WW_R - 1 \rightarrow WW_R$	·	·	·	·	·	·
	DEC IX	11 011 101	2	7	$IX_R - 1 \rightarrow IX_R$	·	·	·	·	·	·
	DEC IY	00 101 011 11 111 101 00 101 011	2	7	$IY_R - 1 \rightarrow IY_R$	·	·	·	·	·	·
INC	INC WW	00 WW 0 011	1	4	$WW_R + 1 \rightarrow WW_R$	·	·	·	·	·	·
	INC IX	11 011 101	2	7	$IX_R + 1 \rightarrow IX_R$	·	·	·	·	·	·
	INC IY	00 100 011 11 111 101 00 100 011	2	7	$IY_R + 1 \rightarrow IY_R$	·	·	·	·	·	·
SBC	SBC HL, WW	11 101 101 01 WW 0 010	2	10	$HL_R - WW_R - C \rightarrow HL_R$	↓	↓	X	V	S	↓

付 録

(2) 転送命令

1. 8ビット転送

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Load 8-bit Data	LD A, I	11 101 101 01 010 111	2	6	Ir→Ar ①	↑	↑	R	IEF ₂	R	.
	LD A, R	11 101 101 01 011 111	2	6	Rr→Ar ①	↑	↑	R	IEF ₂	R	.
	LD A, (BC)	00 001 010	1	6	(BC) _M →Ar
	LD A, (DE)	00 011 010	1	6	(DE) _M →Ar
	LD A, (mn)	00 111 010 <n> <m>	3	12	(mn) _M →Ar
	LD I, A	11 101 101 01 000 111	2	6	Ar→Ir
	LD R, A	11 101 101 01 001 111	2	6	Ar→Rr
	LD (BC), A	00 000 010	1	7	Ar→(BC) _M
	LD (DE), A	00 010 110	1	7	Ar→(DE) _M
	LD (mn), A	00 110 010 <n> <m>	1	13	Ar→(mn) _M
	LD g, g'	01 g g'	1	4	gr→gr
	LD g, (HL)	01 g 110	1	6	(HL) _M →gr
	LD g, m	00 g 110 <m>	2	6	m→gr
	LD g, (IX+d)	11 011 101 01 g 110 <d>	3	14	(IX+d) _M →gr
	LD g, (IY+d)	11 111 101 01 g 110 <d>	3	14	(IY+d) _M →gr
	LD (HL), m	00 110 110 <m>	2	9	m→(HL) _M
	LD (IX+d), m	11 011 101 00 110 110 <d> <m>	4	15	m→(IX+d) _M
	LD (IY+d), m	11 111 101 00 110 110 <d> <m>	4	15	m→(IY+d) _M
	LD (HL), g	01 110 g	1	7	gr→(HL) _M
	LD (IX+d), g	11 011 101 01 110 g <d>	3	15	gr→(IX+d) _M
	LD (IY+d), g	11 111 101 01 110 g <d>	3	15	gr→(IY+d) _M

① LD A, I または LD A, R の命令の最後では、割込みはサンプルされません。

2. 16 ビット転送

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Load 16-bit Data	LD ww, mn	00 ww0 001 <n> <m>	3	9	mn → ww _R
	LD IX, mn	11 011 101 00 100 001 <n> <m>	4	12	mn → IX _R
	LD IY, mn	11 111 101 00 100 001 <n> <m>	4	12	mn → IY _R
	LD SP, HL	11 111 001	1	4	HL _R → SP _R
	LD SP, IX	11 011 101 11 111 001	2	7	IX _R → SP _R
	LD SP, IY	11 111 101 11 111 001	2	7	IY _R → SP _R
	LD ww, (mn)	11 101 101 01 ww1 011 <n> <m>	4	18	(mn + 1) _M → wwHr (mn) _M → wwLr
	LD HL, (mn)	00 101 010 <n> <m>	3	15	(mn + 1) _M → Hr (mn) _M → Lr
	LD IX, (mn)	11 011 101 00 101 010 <n> <m>	4	18	(mn + 1) _M → IXHr (mn) _M → IXLr
	LD IY, (mn)	11 111 101 00 101 010 <n> <m>	4	18	(mn + 1) _M → IYHr (mn) _M → IYLr
	LD(mn), ww	11 101 101 01 ww0 011 <n> <m>	4	19	wwHr → (mn + 1) _M wwLr → (mn) _M
	LD(mn), HL	00 100 010 <n> <m>	3	16	Hr → (mn + 1) _M Lr → (mn) _M
	LD(mn), IX	11 011 101 00 100 010 <n> <m>	4	19	IXHr → (mn + 1) _M IXLr → (mn) _M
	LD(mn), IY	11 111 101 00 100 010 <n> <m>	4	19	IYHr → (mn + 1) _M IYLr → (mn) _M

3. ブロック転送

オペレー ション名	ニーモニック	オペコード	バイ ト数	ステート数	オペレーション	フ ラ グ						
						7	6	4	2	1	0	
						S	Z	H	P/V	N	C	
Block Transfer Search Data	CPD	11 101 101	2	12	$Ar \leftarrow (HL)_M$	↑	③	↑	↑	↑	S	·
		10 101 001			$BC_R - 1 \rightarrow BC_R$							
	CPDR	11 101 101	2	14	$HL_R - 1 \rightarrow HL_R$		③	↑	↑	↑	S	·
		10 111 001		12	$BC_R \neq 0 \text{ } Ar \neq (HL)_M$ $BC_R = 0 \text{ or } Ar = (HL)_M$	↑	↑	↑	↑	↑	S	·
	CPI	11 101 101	2	12	$Ar \leftarrow (HL)_M$							
		10 100 001			$BC_R - 1 \rightarrow BC_R$							
	CPIR	11 101 101	2	14	$HL_R + 1 \rightarrow HL_R$		③	↑	↑	↑	S	·
		10 110 001		12	$BC_R \neq 0 \text{ } Ar \neq (HL)_M$ $BC_R = 0 \text{ or } Ar = (HL)_M$	↑	↑	↑	↑	↑	S	·
	LDD	11 101 101	2	12	$Ar \leftarrow (HL)_M$							
		10 101 000			$Q \left\{ \begin{array}{l} BC_R - 1 \rightarrow BC_R \\ HL_R - 1 \rightarrow HL_R \end{array} \right.$ Repeat Q until $Ar = (HL)_M \text{ or } BC_R = 0$							
	LDDR	11 101 101	2	14 $BC_R \neq 0$ 12 $BC_R = 0$	$(HL)_M \rightarrow (DE)_M$							
		10 111 000		$Q \left\{ \begin{array}{l} BC_R - 1 \rightarrow BC_R \\ DE_R - 1 \rightarrow DE_R \\ HL_R - 1 \rightarrow HL_R \end{array} \right.$ Repeat Q until $BC_R = 0$								
LDI	11 101 101	2	12	$Ar \leftarrow (HL)_M$								
	10 100 000			$BC_R - 1 \rightarrow BC_R$								
LDIR	11 101 101	2	14 $BC_R \neq 0$ 12 $BC_R = 0$	$HL_R + 1 \rightarrow HL_R$								
	10 110 000		$Q \left\{ \begin{array}{l} (HL)_M \rightarrow (DE)_M \\ BC_R - 1 \rightarrow BC_R \\ DE_R + 1 \rightarrow DE_R \\ HL_R + 1 \rightarrow HL_R \end{array} \right.$ Repeat Q until $BC_R = 0$									

② $P/V = 0 : BC_R - 1 = 0$
 $P/V = 1 : BC_R - 1 \neq 0$

③ $Z = 1 : Ar = (HL)_M$
 $Z = 0 : Ar \neq (HL)_M$

4. スタック処理および交換

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
PUSH	PUSH zz	11 zz0 101	1	11	zzLr \rightarrow (SP-2) _M zzHr \rightarrow (SP-1) _M SP _R -2 \rightarrow SP _R
	PUSH IX	11 011 101 11 100 101	2	14	IXLr \rightarrow (SP-2) _M IXHr \rightarrow (SP-1) _M SP _R -2 \rightarrow SP _R
	PUSH IY	11 111 101 11 100 101	2	14	IYLr \rightarrow (SP-2) _M IYHr \rightarrow (SP-1) _M SP _R -2 \rightarrow SP _R
POP	POP zz	11 zz0 001	1	9	(SP+1) _M \rightarrow zzHr ④ (SP) _M \rightarrow zzLr SP _R +2 \rightarrow SP _R
	POP IX	11 011 101 11 100 001	2	12	(SP+1) _M \rightarrow IXHr (SP) _M \rightarrow IXLr SP _R +2 \rightarrow SP _R
	POP IY	11 111 101 11 100 001	2	12	(SP+1) _M \rightarrow IYHr (SP) _M \rightarrow IYLr SP _R +2 \rightarrow SP _R
Exchange	EX AF, AF'	00 001 000	1	4	AF _R \leftrightarrow AF' _R
	EX DE, HL	11 101 011	1	3	DE _R \leftrightarrow HL _R
	EXX	11 011 001	1	3	BC _R \leftrightarrow BC' _R DE _R \leftrightarrow DE' _R HL _R \leftrightarrow HL' _R
	EX (SP), HL	11 100 011	1	16	Hr \leftrightarrow (SP+1) _M Lr \leftrightarrow (SP) _M
	EX (SP), IX	11 011 101 11 100 011	2	19	IXHr \leftrightarrow (SP+1) _M IXLr \leftrightarrow (SP) _M
	EX (SP), IY	11 111 101 11 100 011	2	19	IYHr \leftrightarrow (SP+1) _M IYLr \leftrightarrow (SP) _M

④ POP AF では、スタックの内容がフラグに書き込まれます。

(3) プログラム制御命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Call	CALL mn	11 001 101 <n> <m>	3	16	PCHr→(SP-1) _M PCLr→(SP-2) _M mn→PC _R SP _R -2→SP _R
	CALL f, mn	11 f 100 <n> <m>	3	6(f:false) 16(f:true)	continue:f is false CALL mn:f is true
Jump	DJNZ j	00 010 000 <j-2>	2 2	9(Br≠0) 7(Br=0)	Br-1→Br continue:Br=0
	JP f, mn	11 f 010 <n> <m>	3 3	6(f:false) 9(f:true)	PC _R +j→PC _R :Br≠0 mn→PC _R :f is true continue:f is false
	JP mn	11 000 011 <n> <m>	3	9	mn→PC _R
	JP (HL)	11 101 001	1	3	HL _R →PC _R
	JP (IX)	11 011 101	2	6	IX _R →PC _R
	JP (IY)	11 101 001 11 111 101 11 101 001	2	6	IY _R →PC _R
	JR j	00 011 000 <j-2>	2	8	PC _R +j→PC _R
	JR C, j	00 111 000 <j-2>	2 2	6 8	continue:C=0 PC _R +j→PC _R :C=1
	JR NC, j	00 110 000 <j-2>	2 2	6 8	continue:C=1 PC _R +j→PC _R :C=0
	JR Z, j	00 101 000 <j-2>	2 2	6 8	continue:Z=0 PC _R +j→PC _R :Z=1
	JR NZ, j	00 100 000 <j-2>	2 2	6 8	continue:Z=1 PC _R +j→PC _R :Z=0
Return	RET	11 001 001	1	9	(SP) _M →PCLr (SP+1) _M →PCHr SP _R +2→SP _R
	RET f	11 f 000	1	5(f:false) 10(f:true)	continue:f is false RET:f is true
	RETI	11 101 101 01 001 101	2	12(RI) 22(Z)	(SP) _M →PCLr (SP+1) _M →PCHr SP _R +2→SP _R
	RETN	11 101 101 01 000 101	2	12	(SP) _M →PCLr (SP+1) _M →PCHr SP _R +2→SP _R IEF ₂ →IEF ₁

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Restart	RST v	11 v 111	1	11	PCHr \rightarrow (SP-1) _M PCLr \rightarrow (SP-2) _M 0 \rightarrow PCHr v \rightarrow PCLr SP _R -2 \rightarrow SP _R

(4) I/O 命 令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
INPUT	IN A, (m)	11 011 011 (m)	2	9	(Am) ₁ \rightarrow Ar m \rightarrow A ₀ ~A ₇ Ar \rightarrow A ₈ ~A ₁₅
	IN g, (C)	11 101 101 01 g 000	2	9	(BC) ₁ \rightarrow gr g=110: Only the flags will change. Cr \rightarrow A ₀ ~A ₇ Br \rightarrow A ₈ ~A ₁₅	↑	↑	R	P	R	.
	INO g, (m)	11 101 101 00 g 000 (m)	3	12	(00m) _X \rightarrow gr g=110: Only the flags will change. m \rightarrow A ₀ ~A ₇ 00 \rightarrow A ₈ ~A ₁₅	↑	↑	R	P	R	.
	IND	11 101 101 10 101 010	2	12	(BC) ₁ \rightarrow (HL) _M HL _R -1 \rightarrow HL _R Br-1 \rightarrow Br Cr \rightarrow A ₀ ~A ₇ Br \rightarrow A ₈ ~A ₁₅	X	⑤ ↑	X	X	⑥ ↑	X
	INDR	11 101 101 10 111 010	2	14 (Br \neq 0) 12 (Br=0)	Q (BC) ₁ \rightarrow (HL) _M HL _R -1 \rightarrow HL _R Br-1 \rightarrow Br Repeat Q until Br=0 Cr \rightarrow A ₀ ~A ₇ Br \rightarrow A ₈ ~A ₁₅	X	S	X	X	⑥ ↑	X
	INI	11 101 101 10 100 010	2	12	(BC) ₁ \rightarrow (HL) _M HL _R +1 \rightarrow HL _R Br-1 \rightarrow Br	X	⑤ ↑	X	X	⑥ ↑	X

⑤ Z=1: Br-1=0
Z=0: Br-1 \neq 0

⑥ N=1: MSB of Data=1
N=0: MSB of Data=0

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
	INIR	11 101 101 10 110 010	2	14 (Br≠0) 12 (Br=0)	Cr→A ₀ ~A ₇ Br→A ₈ ~A ₁₅ Q (BC) ₁ →(HL) _M HL _R +1→HL _R Br-1→Br Repeat Q until Br=0 Cr→A ₀ ~A ₇ Br→A ₈ ~A ₁₅	X	S	X	X	⑥ ↑	X
OUTPUT	OUT(m), A	11 010 011 (m)	2	10	Ar→(Am) ₁ m→A ₀ ~A ₇ Ar→A ₈ ~A ₁₅
	OUT(C), g	11 101 101 01 g 001	3	10	gr→(BC) ₁ Cr→A ₀ ~A ₇ Br→A ₈ ~A ₁₅
	OUTO(m), g	11 101 101 00 g 001 (m)	3	13	gr→(00m) ₁ m→A ₀ ~A ₇ 00→A ₈ ~A ₁₅
	OTDM	11 101 101 10 001 011	2	14	(HL) _M →(00C) ₁ HL _R -1→HL _R Cr-1→Cr Br-1→Br Cr→A ₀ ~A ₇ 00→A ₈ ~A ₁₅	↑	⑤ ↑	↑	P	⑥ ↑	↑
	OTDMR	11 101 101 10 011 011	2	16 (Br≠0) 14 (Br=0)	(HL) _M →(00C) ₁ Q HL _R -1→HL _R Cr-1→Cr Br-1→Br Repeat Q until Br=0 Cr→A ₀ ~A ₇ 00→A ₈ ~A ₁₅	R	S	R	S	⑥ ↑	R
	OTDR	11 101 101 10 111 011	2	14 (Br≠0) 12 (Br=0)	(HL) _M →(BC) ₁ Q HL _R -1→HL _R Br-1→Br Repeat Q until Br=0 Cr→A ₀ ~A ₇ 00→A ₈ ~A ₁₅	X	S	X	X	⑥ ↑	X
	OUTI	11 101 101 10 100 011	2	12	Cr→A ₀ ~A ₇ Br→A ₈ ~A ₁₅ (HL) _M →(BC) ₁ HL _R +1→HL _R Br-1→Br Cr→A ₀ ~A ₇	X	⑤ ↑	X	X	⑥ ↑	X

⑤ Z=1 : Br-1=0
Z=0 : Br-1≠0

⑥ N=1 : MSB of Data=1
N=0 : MSB of Data=0

(次頁に続く)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
	OTIR	11 101 101 10 110 011	2	14 (Br≠0) 12 (Br=0)	Br → A ₈ ~ A ₁₅ Q (HL) _M → (BC) ₁ HL _R + 1 → HL _R Br - 1 → Br Repeat Q until Br = 0 Cr → A ₀ ~ A ₇ Br → A ₈ ~ A ₁₅	X	S	X	X	⑥ ↑	X
	TSTIO m	11 101 101 01 110 100 (m)	3	12	(00C) ₁ · m Cr → A ₀ ~ A ₇ 00 → A ₈ ~ A ₁₅	↑	↑	S	P	R	R
	OTIM	11 101 101 10 000 011	2	14	(HL) _M → (00C) ₁ HL _R + 1 → HL _R Cr + 1 → Cr Br - 1 → Br Cr → A ₀ ~ A ₇ 00 → A ₈ ~ A ₁₅	↑	↑	↑	P	↑	↑
	OTIMR	11 101 101 10 010 011	2	16 (Br≠0) 14 (Br=0)	Q (HL) _M → (00C) ₁ HL _R + 1 → HL _R Cr + 1 → Cr Br - 1 → Br Repeat Q until Br = 0 Cr → A ₀ ~ A ₇ 00 → A ₈ ~ A ₁₅	R	S	R	S	⑥ ↑	R
	OUTD	11 101 101 10 101 011	2	12	(HL) _M → (BC) ₁ HL _R - 1 → HL _R Br - 1 → Br Cr → A ₀ ~ A ₇ Br → A ₈ ~ A ₁₅	X	⑤ ↑	X	X	⑥ ↑	X

⑤ Z = 1 : Br - 1 = 0
Z = 0 : Br - 1 ≠ 0

⑥ N = 1 : MSB of Data = 1
N = 0 : MSB of Data = 0

(5) 特殊制御命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フ ラ グ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Special Function	DAA	00 100 111	1	4	Decimal Adjust Accumulator	↑	↑	↑	P	·	↑
Carry Control	CCF	00 111 111	1	3	$\bar{C} \rightarrow C$	·	·	R	·	R	↑
	SCF	00 110 111	1	3	$1 \rightarrow C$	·	·	R	·	R	S
CPU Control	DI	11 110 011	1	3	$0 \rightarrow IEF_1, 0 \rightarrow IEF_2$ ⑦	·	·	·	·	·	·
	EI	11 111 011	1	3	$1 \rightarrow IEF_1, 1 \rightarrow IEF_2$ ⑦	·	·	·	·	·	·
	HALT	01 110 110	1	3	CPU halted	·	·	·	·	·	·
	IM 0	11 101 101	2	6	Interrupt mode 0	·	·	·	·	·	·
		01 000 110									
	IM 1	11 101 101	2	6	Interrupt mode 1	·	·	·	·	·	·
		01 010 110									
	IM 2	11 101 101	2	6	Interrupt mode 2	·	·	·	·	·	·
		01 011 110									
	NOP	00 000 000	1	3	No operation	·	·	·	·	·	·
SLP	SLP	11 101 101	2	8	Sleep	·	·	·	·	·	·
		01 110 110									

⑦ DI または EI 命令の最後では、割込みはサンプルされません。

2 内蔵 I/O レジスタ一覧

I/O レジスタのアドレスは、上位 8 ビットすべて“0”であり、下位 8 ビット中 MSB から 2 ビットは、I/O コントロールレジスタ内の IOA7 と IOA6 により設定できます、以下の表に示すアドレスは、IOA7 と IOA6 が“0”の場合です。

レジスタ	アド レス	備 考																											
ASCIコントロール レジスタ A チャンネル0: CNTLA0	00	<table><tr><td>ビット</td><td>MPE</td><td>RE</td><td>TE</td><td>RTS0</td><td>MPBR EFR</td><td>MOD2</td><td>MOD1</td><td>MOD0</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>0</td><td>1</td><td>不定</td><td>0</td><td>0</td><td>0</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div><div>Multi Processor Enable</div><div>Receive Enable</div><div>Transmit Enable</div><div>Request To Send</div><div>Multi Processor Bit Receive/ Error Flag Reset</div><div>MODE Selection</div></div>	ビット	MPE	RE	TE	RTS0	MPBR EFR	MOD2	MOD1	MOD0	リセット時	0	0	0	1	不定	0	0	0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	ビット	MPE	RE	TE	RTS0	MPBR EFR	MOD2	MOD1	MOD0																				
リセット時	0	0	0	1	不定	0	0	0																					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W																					
ASCIコントロール レジスタ A チャンネル1: CNTLA1	01	<table><tr><td>ビット</td><td>MPE</td><td>RE</td><td>TE</td><td>CKA1D</td><td>MPBR EFR</td><td>MOD2</td><td>MOD1</td><td>MOD0</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>0</td><td>0</td><td>不定</td><td>0</td><td>0</td><td>0</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div><div>Multi Processor Enable</div><div>Receive Enable</div><div>Transmit Enable</div><div>CKA1 Disable</div><div>Multi Processor Bit Receive/ Error Flag Reset</div><div>MODE Selection</div></div> <div>MOD2, 1, 0</div> <div>0 0 0 Start + 7bit Data + 1stop</div> <div>0 0 1 Start + 7bit Data + 2stop</div> <div>0 1 0 Start + 7bit Data + Parity + 1stop</div> <div>0 1 1 Start + 7bit Data + Parity + 2stop</div> <div>1 0 0 Start + 8bit Data + 1stop</div> <div>1 0 1 Start + 8bit Data + 2stop</div> <div>1 1 0 Start + 8bit Data + Parity + 1stop</div> <div>1 1 1 Start + 8bit Data + Parity + 2stop</div>	ビット	MPE	RE	TE	CKA1D	MPBR EFR	MOD2	MOD1	MOD0	リセット時	0	0	0	0	不定	0	0	0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ビット	MPE	RE	TE	CKA1D	MPBR EFR	MOD2	MOD1	MOD0																					
リセット時	0	0	0	0	不定	0	0	0																					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W																					

レジスタ	アド レス	備 考																																																																													
ASCIコントロール レジスタB チャンネル0: CNTLB0	02	<table><tr><td>ビット</td><td>MPBT</td><td>MP</td><td>CTS PS</td><td>PEO</td><td>DR</td><td>SS2</td><td>SS1</td><td>SS0</td></tr><tr><td>リセット時</td><td>不 定</td><td>0</td><td>*</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div><div>└ Multi Processor Bit Transmit</div><div>└ Multi Processor</div><div>└ Clear To Send/Prescale</div><div>└ Parity Even or Odd</div><div>└ Divide Ratio</div><div>└ Clock Source and Speed Select</div></div>	ビット	MPBT	MP	CTS PS	PEO	DR	SS2	SS1	SS0	リセット時	不 定	0	*	0	0	1	1	1	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W																																																		
ビット	MPBT	MP	CTS PS	PEO	DR	SS2	SS1	SS0																																																																							
リセット時	不 定	0	*	0	0	1	1	1																																																																							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W																																																																							
ASCIコントロール レジスタB チャンネル1: CNTLB1	03	<table><tr><td>ビット</td><td>MPBT</td><td>MP</td><td>CTS PS</td><td>PEO</td><td>DR</td><td>SS2</td><td>SS1</td><td>SS0</td></tr><tr><td>リセット時</td><td>不 定</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div><div>└ Multi Processor Bit Transmit</div><div>└ Multi Processor</div><div>└ Clear To Send/Prescale</div><div>└ Parity Even or Odd</div><div>└ Divide Ratio</div><div>└ Clock Source and Speed Select</div></div> <table><tr><td>総合分周比</td><td colspan="2">PS=0 (分周比=10)</td><td colspan="2">PS=1 (分周比=30)</td></tr><tr><td>SS 2. 1. 0</td><td>DR=0(×16)</td><td>DR=1(×64)</td><td>DR=0(×16)</td><td>DR=1(×64)</td></tr><tr><td>0 0 0</td><td>φ÷ 160</td><td>φ÷ 640</td><td>φ÷ 480</td><td>φ÷ 1 920</td></tr><tr><td>0 0 1</td><td>÷ 320</td><td>÷ 1 280</td><td>÷ 960</td><td>÷ 3 840</td></tr><tr><td>0 1 0</td><td>÷ 640</td><td>÷ 2 560</td><td>÷ 1 920</td><td>÷ 7 680</td></tr><tr><td>0 1 1</td><td>÷ 1 280</td><td>÷ 5 120</td><td>÷ 3 840</td><td>÷ 15 360</td></tr><tr><td>1 0 0</td><td>÷ 2 560</td><td>÷ 10 240</td><td>÷ 7 680</td><td>÷ 30 720</td></tr><tr><td>1 0 1</td><td>÷ 5 120</td><td>÷ 20 480</td><td>÷ 15 360</td><td>÷ 61 440</td></tr><tr><td>1 1 0</td><td>÷ 10 240</td><td>÷ 40 960</td><td>÷ 30 720</td><td>÷ 122 880</td></tr><tr><td>1 1 1</td><td colspan="4">外部クロック入力 (÷40以上)</td></tr></table>	ビット	MPBT	MP	CTS PS	PEO	DR	SS2	SS1	SS0	リセット時	不 定	0	0	0	0	1	1	1	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	総合分周比	PS=0 (分周比=10)		PS=1 (分周比=30)		SS 2. 1. 0	DR=0(×16)	DR=1(×64)	DR=0(×16)	DR=1(×64)	0 0 0	φ÷ 160	φ÷ 640	φ÷ 480	φ÷ 1 920	0 0 1	÷ 320	÷ 1 280	÷ 960	÷ 3 840	0 1 0	÷ 640	÷ 2 560	÷ 1 920	÷ 7 680	0 1 1	÷ 1 280	÷ 5 120	÷ 3 840	÷ 15 360	1 0 0	÷ 2 560	÷ 10 240	÷ 7 680	÷ 30 720	1 0 1	÷ 5 120	÷ 20 480	÷ 15 360	÷ 61 440	1 1 0	÷ 10 240	÷ 40 960	÷ 30 720	÷ 122 880	1 1 1	外部クロック入力 (÷40以上)			
ビット	MPBT	MP	CTS PS	PEO	DR	SS2	SS1	SS0																																																																							
リセット時	不 定	0	0	0	0	1	1	1																																																																							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W																																																																							
総合分周比	PS=0 (分周比=10)		PS=1 (分周比=30)																																																																												
SS 2. 1. 0	DR=0(×16)	DR=1(×64)	DR=0(×16)	DR=1(×64)																																																																											
0 0 0	φ÷ 160	φ÷ 640	φ÷ 480	φ÷ 1 920																																																																											
0 0 1	÷ 320	÷ 1 280	÷ 960	÷ 3 840																																																																											
0 1 0	÷ 640	÷ 2 560	÷ 1 920	÷ 7 680																																																																											
0 1 1	÷ 1 280	÷ 5 120	÷ 3 840	÷ 15 360																																																																											
1 0 0	÷ 2 560	÷ 10 240	÷ 7 680	÷ 30 720																																																																											
1 0 1	÷ 5 120	÷ 20 480	÷ 15 360	÷ 61 440																																																																											
1 1 0	÷ 10 240	÷ 40 960	÷ 30 720	÷ 122 880																																																																											
1 1 1	外部クロック入力 (÷40以上)																																																																														

レジスタ	アド レス	備 考																																				
ASCIステータス レジスタ チャンネル0: STAT0	04	<table><tr><td>ビット</td><td>RDRF</td><td>OVRN</td><td>PE</td><td>FE</td><td>RIE</td><td>DCD0</td><td>TDRE</td><td>TIE</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>*</td><td>**</td><td>0</td></tr><tr><td>R/W</td><td>R</td><td>R</td><td>R</td><td>R</td><td>R/W</td><td>R</td><td>R</td><td>R/W</td></tr></table> <div><div>Receive Data Register Full</div><div>Over Run Error</div><div>Parity Error</div><div>Framing Error</div><div>Receive Interrupt Enable</div><div>Data Carrier Detect</div><div>Transmit Data Register Empty</div><div>Transmit Interrupt Enable</div></div> <p>* 端子の状態を取り込みます</p> <table><tr><td>**</td><td>CTS₀ 端子</td><td>TDRE</td></tr><tr><td></td><td>L</td><td>1</td></tr><tr><td></td><td>H</td><td>0</td></tr></table>	ビット	RDRF	OVRN	PE	FE	RIE	DCD0	TDRE	TIE	リセット時	0	0	0	0	0	*	**	0	R/W	R	R	R	R	R/W	R	R	R/W	**	CTS ₀ 端子	TDRE		L	1		H	0
ビット	RDRF	OVRN	PE	FE	RIE	DCD0	TDRE	TIE																														
リセット時	0	0	0	0	0	*	**	0																														
R/W	R	R	R	R	R/W	R	R	R/W																														
**	CTS ₀ 端子	TDRE																																				
	L	1																																				
	H	0																																				
ASCIステータス レジスタ チャンネル1: STAT1	05	<table><tr><td>ビット</td><td>RDRF</td><td>OVRN</td><td>PE</td><td>FE</td><td>RIE</td><td>CTS1E</td><td>TDRE</td><td>TIE</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>R/W</td><td>R</td><td>R</td><td>R</td><td>R</td><td>R/W</td><td>R/W</td><td>R</td><td>R/W</td></tr></table> <div><div>Receive Data Register Full</div><div>Over Run Error</div><div>Parity Error</div><div>Framing Error</div><div>Receive Interrupt Enable</div><div>CTS₁ Enable</div><div>Transmit Data Register Empty</div><div>Transmit Interrupt Enable</div></div>	ビット	RDRF	OVRN	PE	FE	RIE	CTS1E	TDRE	TIE	リセット時	0	0	0	0	0	0	1	0	R/W	R	R	R	R	R/W	R/W	R	R/W									
ビット	RDRF	OVRN	PE	FE	RIE	CTS1E	TDRE	TIE																														
リセット時	0	0	0	0	0	0	1	0																														
R/W	R	R	R	R	R/W	R/W	R	R/W																														
ASCIトランス ミットデータ レジスタチャンネル 0: TDR0	06																																					
ASCIトランス ミットデータ レジスタチャンネル 1: TDR1	07																																					

レジスタ	アド レス	備 考																																												
ASCIIレシーブ データレジスタ チャンネル0: RDRO	08																																													
ASCIIレシーブ データレジスタ チャンネル1: RDR1	09																																													
CSI/Oコント ロールレジスタ : CNTR	0A	<div>ビット</div> <table><tr><td>EF</td><td>EIE</td><td>RE</td><td>TE</td><td>—</td><td>SS2</td><td>SS1</td><td>SS0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>R</td><td>R/W</td><td>R/W</td><td>R/W</td><td></td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div>リセット時</div> <div>R/W</div> <div>Speed Select</div> <div>Transmit Enable</div> <div>Receive Enable</div> <div>End Interrupt Enable</div> <div>End Flag</div> <table><tr><td>SS2. 1. 0</td><td>ボーレート</td><td>SS2. 1. 0</td><td>ボーレート</td></tr><tr><td>0 0 0</td><td>$\phi \div 20$</td><td>1 0 0</td><td>$\phi \div 320$</td></tr><tr><td>0 0 1</td><td>$\div 40$</td><td>1 0 1</td><td>$\div 640$</td></tr><tr><td>0 1 0</td><td>$\div 80$</td><td>1 1 0</td><td>$\div 1280$</td></tr><tr><td>0 1 1</td><td>$\div 160$</td><td>1 1 1</td><td>外部($\div 20$以上)</td></tr></table>	EF	EIE	RE	TE	—	SS2	SS1	SS0	0	0	0	0	1	1	1	1	R	R/W	R/W	R/W		R/W	R/W	R/W	SS2. 1. 0	ボーレート	SS2. 1. 0	ボーレート	0 0 0	$\phi \div 20$	1 0 0	$\phi \div 320$	0 0 1	$\div 40$	1 0 1	$\div 640$	0 1 0	$\div 80$	1 1 0	$\div 1280$	0 1 1	$\div 160$	1 1 1	外部($\div 20$ 以上)
EF	EIE	RE	TE	—	SS2	SS1	SS0																																							
0	0	0	0	1	1	1	1																																							
R	R/W	R/W	R/W		R/W	R/W	R/W																																							
SS2. 1. 0	ボーレート	SS2. 1. 0	ボーレート																																											
0 0 0	$\phi \div 20$	1 0 0	$\phi \div 320$																																											
0 0 1	$\div 40$	1 0 1	$\div 640$																																											
0 1 0	$\div 80$	1 1 0	$\div 1280$																																											
0 1 1	$\div 160$	1 1 1	外部($\div 20$ 以上)																																											
CSI/Oトランス ミット/レシーブ データレジスタ: TRDR	0B																																													
タイマデータ レジスタ チャンネル0L: TMDROL	0C																																													
タイマデータ レジスタ チャンネル0H: TMDROH	0D																																													
タイマリロード レジスタ チャンネル0L: RLDROL	0E																																													
タイマリロード レジスタ チャンネル0H: RLDROH	0F																																													

レジスタ	アド レス	備 考																																					
タイマコント ロールレジスタ： TCR	10	<table><tr><td>ビット</td><td>TIF1</td><td>TIF0</td><td>TIE1</td><td>TIE0</td><td>TOC1</td><td>TOC0</td><td>TDE1</td><td>TDE0</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>R/W</td><td>R</td><td>R</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div><div>Timer Interrupt Flag 1. 0</div><div>Timer Interrupt Enable 1. 0</div><div>Timer Output Control 1. 0</div><div>Timer Down Count Enable 1. 0</div></div> <table><tr><td>TOC 1. 0</td><td>A₁₈/TOUT 端子</td></tr><tr><td>0 0</td><td>タイマ出力禁止</td></tr><tr><td>0 1</td><td>トグル出力</td></tr><tr><td>1 0</td><td>"0" を出力</td></tr><tr><td>1 1</td><td>"1" を出力</td></tr></table>	ビット	TIF1	TIF0	TIE1	TIE0	TOC1	TOC0	TDE1	TDE0	リセット時	0	0	0	0	0	0	0	0	R/W	R	R	R/W	R/W	R/W	R/W	R/W	R/W	TOC 1. 0	A ₁₈ /TOUT 端子	0 0	タイマ出力禁止	0 1	トグル出力	1 0	"0" を出力	1 1	"1" を出力
ビット	TIF1	TIF0	TIE1	TIE0	TOC1	TOC0	TDE1	TDE0																															
リセット時	0	0	0	0	0	0	0	0																															
R/W	R	R	R/W	R/W	R/W	R/W	R/W	R/W																															
TOC 1. 0	A ₁₈ /TOUT 端子																																						
0 0	タイマ出力禁止																																						
0 1	トグル出力																																						
1 0	"0" を出力																																						
1 1	"1" を出力																																						
タイマデータ レジスタ チャンネル1L： TMDR1L	14																																						
タイマデータ レジスタ チャンネル1H： TMDR1H	15																																						
タイマリロード レジスタ チャンネル1L： RLDR1L	16																																						
タイマリロード レジスタ チャンネル1H： RLDR1H	17																																						
フリーランニング カウンタ：FRC	18	リードのみ																																					
DMAソース アドレスレジスタ チャンネル0L： SAR0L	20																																						

レジスタ	アド レス	備 考																				
DMAソース アドレスレジスタ チャンネル0H: SAR0H	21																					
DMAソース アドレスレジスタ チャンネル0B: SAR0B	22	ビット 0, 1, 2(, 3*)のみ使用 <table><tr><th>A₁₉*, A₁₈</th><th>A₁₇</th><th>A₁₆</th><th>DMA 転送要求</th></tr><tr><td>×</td><td>×</td><td>0 0</td><td>DREQ₀ (外部)</td></tr><tr><td>×</td><td>×</td><td>0 1</td><td>RDR0 (ASCII0)</td></tr><tr><td>×</td><td>×</td><td>1 0</td><td>RDR1 (ASCII1)</td></tr><tr><td>×</td><td>×</td><td>1 1</td><td>Not Used</td></tr></table>	A ₁₉ *, A ₁₈	A ₁₇	A ₁₆	DMA 転送要求	×	×	0 0	DREQ ₀ (外部)	×	×	0 1	RDR0 (ASCII0)	×	×	1 0	RDR1 (ASCII1)	×	×	1 1	Not Used
A ₁₉ *, A ₁₈	A ₁₇	A ₁₆	DMA 転送要求																			
×	×	0 0	DREQ ₀ (外部)																			
×	×	0 1	RDR0 (ASCII0)																			
×	×	1 0	RDR1 (ASCII1)																			
×	×	1 1	Not Used																			
DMAディスティネ ーションアドレス レジスタチャンネル OL: DAR0L	23																					
DMAディスティネ ーションアドレス レジスタチャンネル OH: DAR0H	24																					
DMAディスティネ ーションアドレス レジスタチャンネル OB: DAR0B	25	ビット 0, 1, 2(, 3*)のみ使用 <table><tr><th>A₁₉*, A₁₈</th><th>A₁₇</th><th>A₁₆</th><th>DMA 転送要求</th></tr><tr><td>×</td><td>×</td><td>0 0</td><td>DREQ₀ (外部)</td></tr><tr><td>×</td><td>×</td><td>0 1</td><td>TDR0 (ASCII0)</td></tr><tr><td>×</td><td>×</td><td>1 0</td><td>TDR1 (ASCII1)</td></tr><tr><td>×</td><td>×</td><td>1 1</td><td>Not Used</td></tr></table>	A ₁₉ *, A ₁₈	A ₁₇	A ₁₆	DMA 転送要求	×	×	0 0	DREQ ₀ (外部)	×	×	0 1	TDR0 (ASCII0)	×	×	1 0	TDR1 (ASCII1)	×	×	1 1	Not Used
A ₁₉ *, A ₁₈	A ₁₇	A ₁₆	DMA 転送要求																			
×	×	0 0	DREQ ₀ (外部)																			
×	×	0 1	TDR0 (ASCII0)																			
×	×	1 0	TDR1 (ASCII1)																			
×	×	1 1	Not Used																			
DMAバイト カウントレジスタ チャンネル0L: BCR0L	26																					
DMAバイト カウントレジスタ チャンネル0H: BCR0H	27																					
DMAメモリ アドレスレジスタ チャンネル1L: MAR1L	28																					
DMAメモリ アドレスレジスタ チャンネル1H: MAR1H	29																					

* CP-68, FP-80のみ有効

レジスタ	アド レス	備 考																											
DMAメモリ アドレスレジスタ チャンネル1B: MAR1B	2A	ビット 0, 1, 2 (, 3*) のみ使用																											
DMAI/Oアドレス レジスタ チャンネル1L: IAR1L	2B																												
DMAI/Oアドレス レジスタ チャンネル1H: IAR1H	2C																												
DMAバイト カウンタレジスタ チャンネル1L: BCR1L	2E																												
DMAバイト カウンタレジスタ チャンネル1H: BCR1H	2F																												
DMAステータス レジスタ: DSTAT	30	<table><tr><td>ビット</td><td>DE1</td><td>DE0</td><td>DWE1</td><td>DWE0</td><td>DIE1</td><td>DIE0</td><td>—</td><td>DME</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td>W</td><td>W</td><td>R/W</td><td>R/W</td><td></td><td>R</td></tr></table> <div><div>DMA Master Enable</div><div>DMA Interrupt Enable 1, 0</div><div>DMA Enable Bit Write Enabl 1, 0</div><div>DMA Enable ch 1, 0</div></div>	ビット	DE1	DE0	DWE1	DWE0	DIE1	DIE0	—	DME	リセット時	0	0	1	1	0	0	1	0	R/W	R/W	R/W	W	W	R/W	R/W		R
ビット	DE1	DE0	DWE1	DWE0	DIE1	DIE0	—	DME																					
リセット時	0	0	1	1	0	0	1	0																					
R/W	R/W	R/W	W	W	R/W	R/W		R																					
DMAモード レジスタ: DMODE	31	<table><tr><td>ビット</td><td>—</td><td>—</td><td>DM1</td><td>DM0</td><td>SM1</td><td>SM0</td><td>MMOD</td><td>—</td></tr><tr><td>リセット時</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>R/W</td><td></td><td></td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td></td></tr></table> <div><div>Memory MODE Select</div><div>ch 0 Source Mode 1, 0</div><div>ch 0 Destination Mode 1, 0</div></div>	ビット	—	—	DM1	DM0	SM1	SM0	MMOD	—	リセット時	1	1	0	0	0	0	0	1	R/W			R/W	R/W	R/W	R/W	R/W	
ビット	—	—	DM1	DM0	SM1	SM0	MMOD	—																					
リセット時	1	1	0	0	0	0	0	1																					
R/W			R/W	R/W	R/W	R/W	R/W																						

レジスタ	アド レス	備 考									
DMA WAIT コントロール レジスタ : DCNTL	32	DM 1, 0		ディスティ ネーション	アドレス	SM 1, 0		ソース	アドレス		
		0 0		M	DAR0+1	0 0		M	SAR0+1		
		0 1		M	DAR0-1	0 1		M	SAR0-1		
		1 0		M	DAR0固定	1 0		M	SAR0固定		
		1 1		I/O	DAR0固定	1 1		I/O	SAR0固定		
		MMOD		モ ー ド							
		0		サイクルスチールモード							
		1		バーストモード							
		ビット		MWI1	MWI0	IWI1	IWI0	DMS1	DMS0	DIM1	DIM0
		リセット時		1	1	1	1	0	0	0	0
R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
		Memory Wait Insertion				DREQi Select, i=1, 0				DMA ch 1 I/O Memory Mode Select	
						I/O Wait Insertion					
		MWI 1, 0		ウェイトステート		IWI 1, 0		ウェイトステート			
		0 0		0		0 0		0			
		0 1		1		0 1		2			
		1 0		2		1 0		3			
		1 1		3		1 1		4			
		DMSi	DREQi入力		DIM 1, 0		転送形態	アドレス増減			
		1	エッジ入力		0 0		M→I/O	MAR1+1 IAR1固定			
		0	レベル入力		0 1		M→I/O	MAR1-1 IAR1固定			
					1 0		I/O→M	IAR1固定 MAR1+1			
					1 1		I/O→M	IAR1固定 MAR1-1			
インタラプトベク タローレジスタ : IL	33	ビット		IL7	IL6	IL5	—	—	—	—	—
		リセット時		0	0	0	0	0	0	0	0
		R/W		R/W	R/W	R/W					
		Interrupt Vector Low									

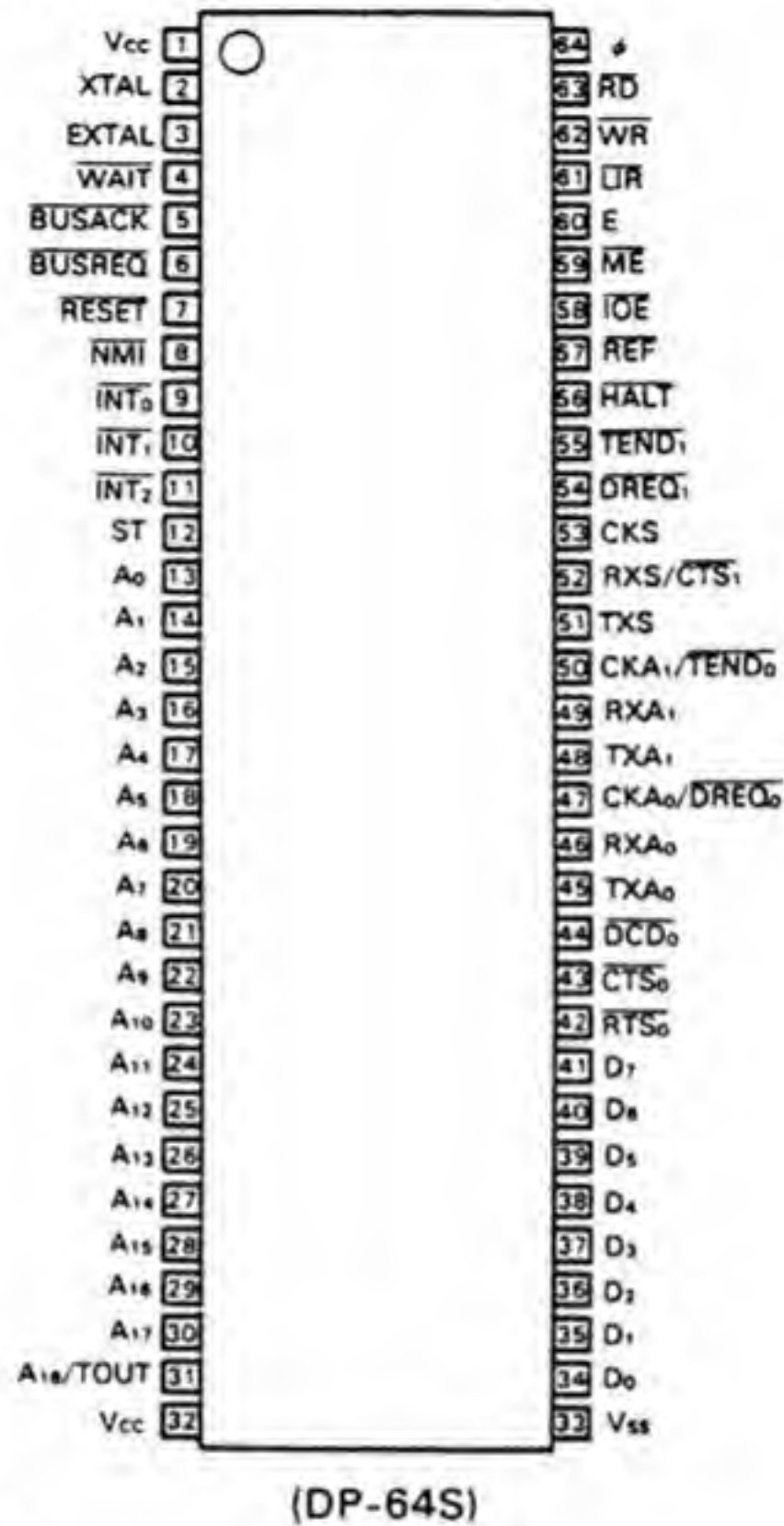
レジスタ	アド レス	備 考																																					
INT/TRAR コントロール レジスタ：ITC	34	<table><tr><td>ビット</td><td>TRAP</td><td>UFO</td><td>—</td><td>—</td><td>—</td><td>ITE2</td><td>ITE1</td><td>ITE0</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>R/W</td><td>R/W</td><td>R</td><td></td><td></td><td></td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div>TRAP</div> <div>Undefined Fetch Object Code</div> <div>INT Enable 2, 1, 0</div>	ビット	TRAP	UFO	—	—	—	ITE2	ITE1	ITE0	リセット時	0	0	1	1	1	0	0	1	R/W	R/W	R				R/W	R/W	R/W										
ビット	TRAP	UFO	—	—	—	ITE2	ITE1	ITE0																															
リセット時	0	0	1	1	1	0	0	1																															
R/W	R/W	R				R/W	R/W	R/W																															
リフレッシュ コントロール レジスタ：RCR	36	<table><tr><td>ビット</td><td>REFE</td><td>REFW</td><td>—</td><td>—</td><td>—</td><td>—</td><td>CYC1</td><td>CYC0</td></tr><tr><td>リセット時</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td></td><td></td><td></td><td></td><td>R/W</td><td>R/W</td></tr></table> <div>Refresh Wait State</div> <div>Refresh Enable</div> <div>Cycle Select</div> <table><tr><td>CYC 1, 0</td><td>リフレッシュサイクル間隔</td></tr><tr><td>0 0</td><td>10 ステート</td></tr><tr><td>0 1</td><td>20</td></tr><tr><td>1 0</td><td>40</td></tr><tr><td>1 1</td><td>80</td></tr></table>	ビット	REFE	REFW	—	—	—	—	CYC1	CYC0	リセット時	1	1	1	1	1	1	0	0	R/W	R/W	R/W					R/W	R/W	CYC 1, 0	リフレッシュサイクル間隔	0 0	10 ステート	0 1	20	1 0	40	1 1	80
ビット	REFE	REFW	—	—	—	—	CYC1	CYC0																															
リセット時	1	1	1	1	1	1	0	0																															
R/W	R/W	R/W					R/W	R/W																															
CYC 1, 0	リフレッシュサイクル間隔																																						
0 0	10 ステート																																						
0 1	20																																						
1 0	40																																						
1 1	80																																						
MMUコモン ベースレジスタ： CBR	38	<table><tr><td>ビット</td><td>CB7*</td><td>CB6</td><td>CB5</td><td>CB4</td><td>CB3</td><td>CB2</td><td>CB1</td><td>CB0</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div>MMU Common Base Register</div>	ビット	CB7*	CB6	CB5	CB4	CB3	CB2	CB1	CB0	リセット時	0	0	0	0	0	0	0	0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W										
ビット	CB7*	CB6	CB5	CB4	CB3	CB2	CB1	CB0																															
リセット時	0	0	0	0	0	0	0	0																															
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W																															
MMUバンク ベースレジスタ： BBR	39	<table><tr><td>ビット</td><td>BB7*</td><td>BB6</td><td>BB5</td><td>BB4</td><td>BB3</td><td>BB2</td><td>BB1</td><td>BB0</td></tr><tr><td>リセット時</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr></table> <div>MMU Bank Base Register</div>	ビット	BB7*	BB6	BB5	BB4	BB3	BB2	BB1	BB0	リセット時	0	0	0	0	0	0	0	0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W										
ビット	BB7*	BB6	BB5	BB4	BB3	BB2	BB1	BB0																															
リセット時	0	0	0	0	0	0	0	0																															
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W																															

* CP-68, FP-80のみ有効

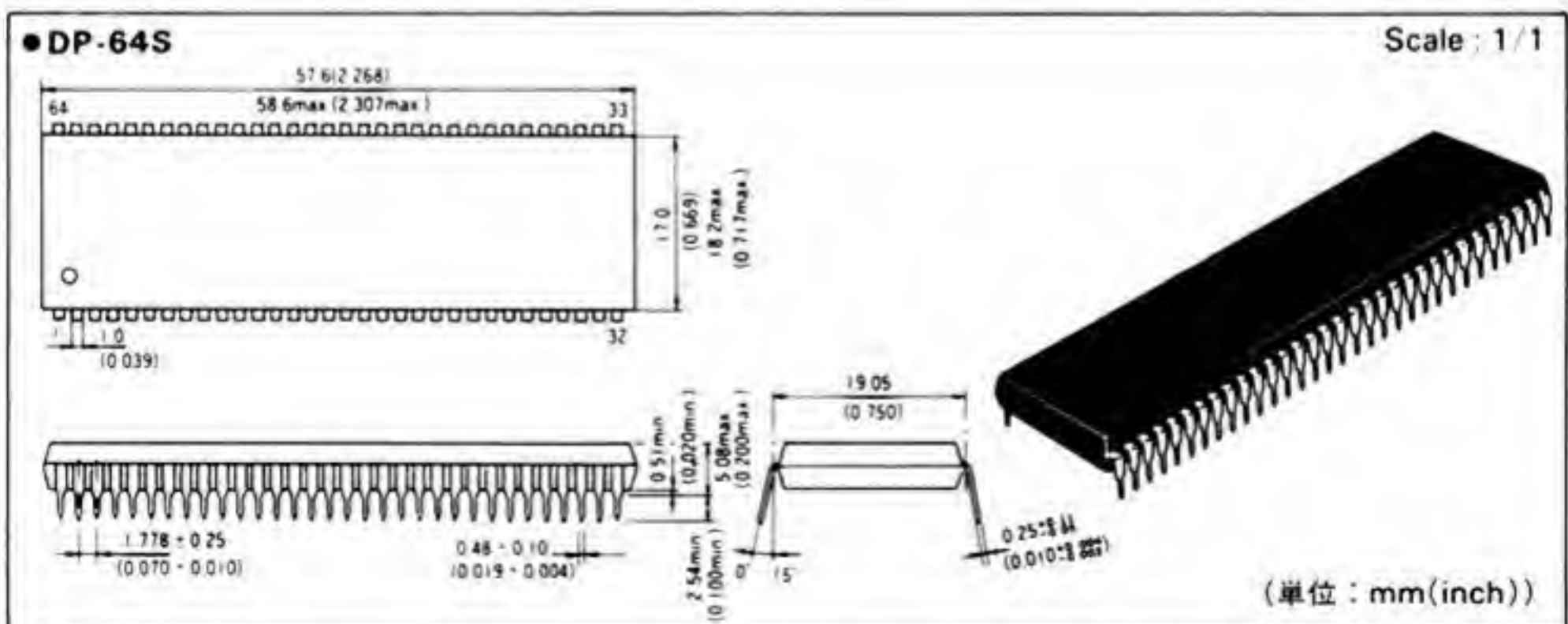
レジスタ	アド レス	備 考								
MMUコモン/ バンクエリア レジスタ：CBAR	3A	ビット	CA3	CA2	CA1	CA0	BA3	BA2	BA1	BA0
		リセット時	1	1	1	1	0	0	0	0
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
			MMU Common Area Register				MMU Bank Area Register			
動作モードコント ロールレジスタ： OMCR (Z版のみ有効)	3E	ビット	LIRE	LIRTE	IOC	—	—	—	—	—
		リセット時	1	1	1	1	1	1	1	1
		R/W	R/W	W	R/W					
			LIR Enable		LIR Temporary Enable		I/O Compatibility			
I/Oコントロール レジスタ：ICR	3F	ビット	IOA7	IOA6	IOSTP	—	—	—	—	—
		リセット時	0	0	0	1	1	1	1	1
		R/W	R/W	R/W	R/W					
			I/O Address		I/O Stop					

3 端子名一覧

1. ピン配置図 (上面図)



2. 外形寸法図

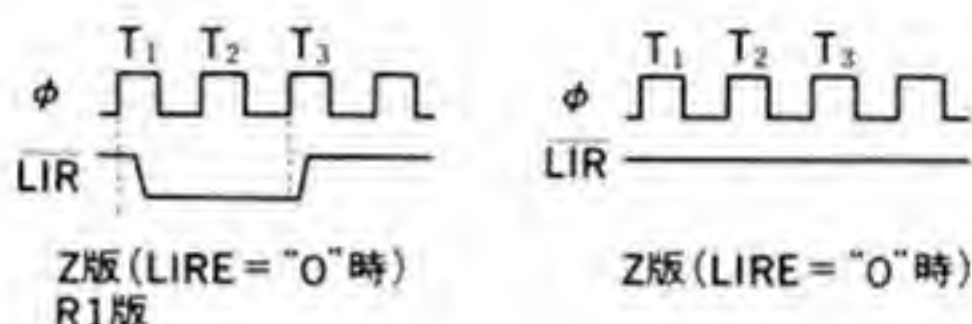


4 R1 版とZ 版との違い

64180Z 版は、64180R1 版に対して上位互換性をもっています。Z 版には次のような機能が追加されています。

1. $\overline{\text{LIR}}$ タイミング

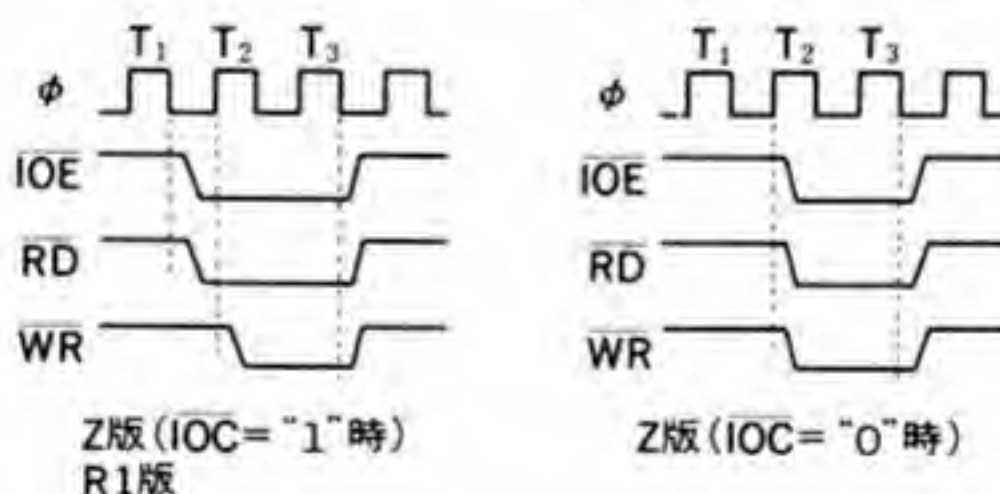
付図1に示すように、オペコードフェッチサイクルでも“H”のままにすることができます。動作モードコントロールレジスタ(OMCR)内のLIREビットに“0”を書き込んだ後の最初のオペコードフェッチから、この状態になります。この状態が必要になるのは、Z80 周辺 LSI をデイジーチェーン割込みで使用する場合があります。



付図1 $\overline{\text{LIR}}$ タイミング

2. $\overline{\text{IOE}}$, $\overline{\text{RD}}$ タイミング

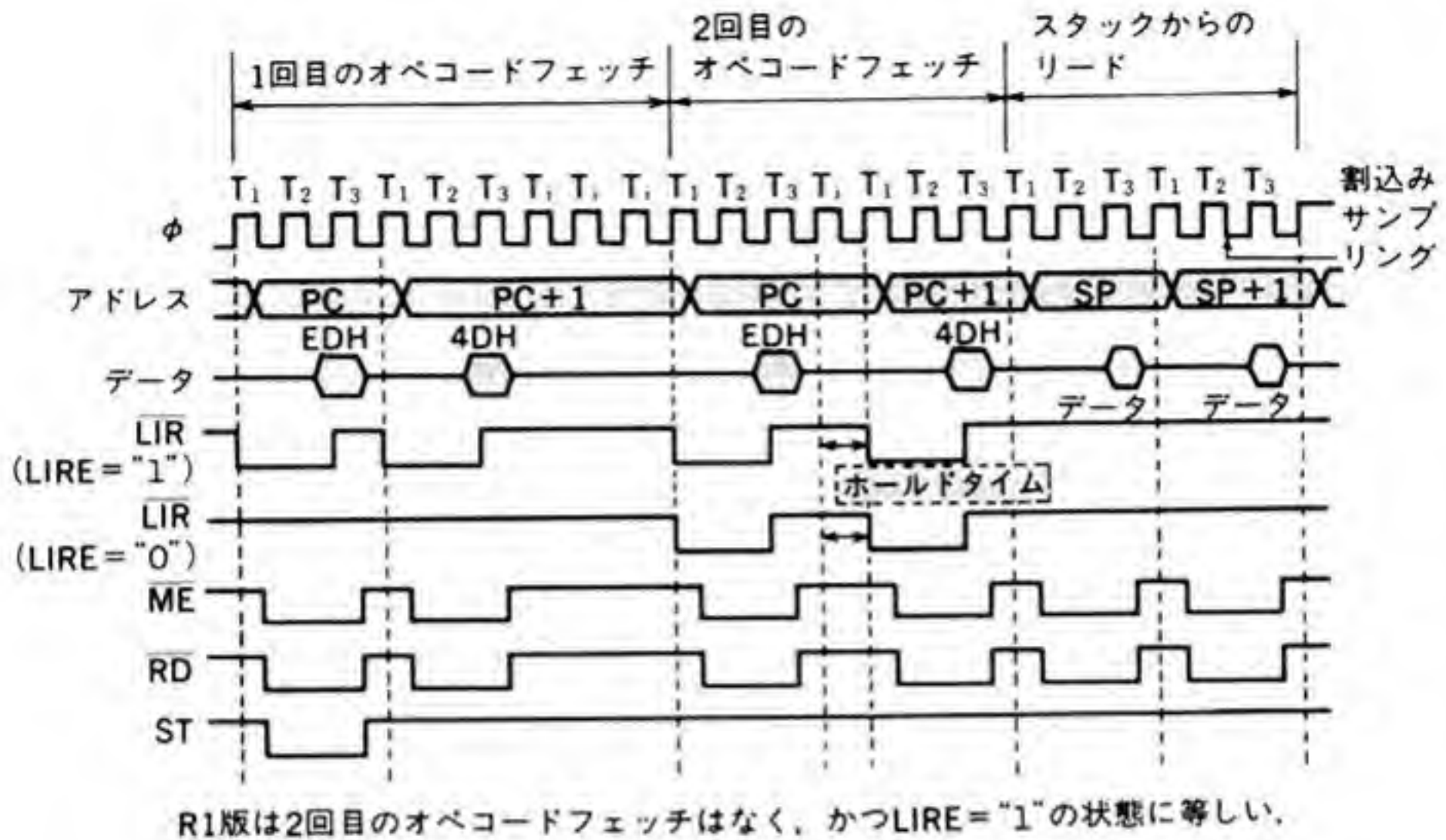
付図2に示すように、 $\overline{\text{IOE}}$ と $\overline{\text{RD}}$ の出力タイミングが OMCR 内の $\overline{\text{IOC}}$ ビットの状態に従って変わります。 $\overline{\text{IOC}}$ ビットに“0”を書き込んだ場合のタイミングでは、Z80 周辺 LSI とのインタフェースが容易になります。



付図2 $\overline{\text{IOE}}$, $\overline{\text{RD}}$ タイミング

3. RETI 命令

この命令(コードは EDH, 4DH)は、Z80 周辺 LSI がデイジーチェーン割込み接続しているときに、割込み要求デバイスが内部の割込み要求をクリアするためにサポートされています。R1 版では、この命令を解読するためのホールドタイムが十分ではありません。Z 版ではこの点を改良し、付図3のように第1 オペコードフェッチサイクルと第2 オペコードフェッチサイクルに1 ステートダミーサイク



付図3 RETI命令の実行タイミング(Z版)

ル T_i を挿入し、解説のためのホールド時間をもたせています。ただし、CPUはこの命令を自動的に2度フェッチしているので、1回目のフェッチ時には $\overline{\text{LIR}}$ はインアクティブにします。この操作は、上記の OMCR の LIRE ビットへ "0" を書き込むことにより行います。付図3のように、2回目のフェッチ時のみ $\overline{\text{LIR}}$ がアクティブとなります。

なお、Z80PIO は内部の割込み回路をイネーブルとするために、 $\overline{\text{M1}}$ 信号の立上りエッジを利用しています。ところが、Z版は LIRE ビットを "0" としたとき、 $\overline{\text{LIR}}$ ($\overline{\text{M1}}$) が出力されません。そこで、 $\overline{\text{LIR}}$ 信号を1回だけ出力させる目的で、Z版は OMCR 内に $\overline{\text{LIRTE}}$ ビットをもっています。 $\overline{\text{LIR}}$ 信号は、 $\overline{\text{LIRTE}}$ ビットへ "0" を書き込んだ後、最初に実行されるオペコードフェッチサイクルで出力されます。ただし、このオペコードフェッチ時のデータバスには、EDH, CBH, FDH, DDH が現れないようにします。すなわち、第2オペコードをもつ命令を実行させないようにします。その具体的なプログラミング例を付図4に示します。

DI	: 割込み受付の禁止
PUSH AF	: アキュムレータの退避
IN0 A, (OMCR)	
RES 6, A	: $\overline{\text{LIRTE}}$ ビットに "0" を書き込む
OUT0 (OMCR), A	
POP AF	: アキュムレータの内容を復帰 (この命令は1バイト命令)
EI	: 割込み受付の許可

付図4 $\overline{\text{LIRTE}}$ ビットへの "0" 書き込み例

5 180 カードパソコンのハードウェア仕様

No.	項 目		仕 様	備 考
1	プロセッサ (MPU)		HD64180R1P6 (6MHz)	HITACHI
2	動作周波数		<ul style="list-style-type: none"> ・ $\phi = 6.144\text{MHz}$ ・ 外部水晶振動子……12.288MHz 	
3	記憶容量 (RAM)	使用メモリ	HM50256-15	
		使用数	8個(すべてデータメモリ)	
		総容量	256K バイト	
	記憶容量 (ROM)	使用メモリ	HN27256G-25 または HN27C64G-25 (HN27C256G-25)/HN: 27128G-25	切替可能
		使用数	1個	
		総容量	32KB/16KB/8KB	
4	シリアルインタフェース		<ul style="list-style-type: none"> ・ RS-232C 通信回線……2ch ……64180内蔵SCI使用 ・ モデム制御信号(RTS, CTS, DCD) 有り ……チャンネル0のみ ・ チャンネル1はターミナル専用……転送レート: 9600bps, ビット長:8ビット, パリティ:なし, ストップビット:2ストップビット 	
5	フロッピーディスクインタフェース		<ul style="list-style-type: none"> ・ 5インチ両面倍密 ・ 専用コントローラ使用……HD63, 265 ・ データセバレータ内蔵 ・ 使用ドライバ可能台数……最大4台まで 	
	ディスクフォーマット		F コマンドでフォーマット可能	モニタプログラム
	記録方式		MFM	
	データ転送		HD64180内蔵DMACによるDMA転送 (デュアルアドレス方式)	
	セクタ長		256 バイト/セクタ	
	対象 F D D		FD-55BV	TEAC
6	フロッピーディスクドライブ(外付)	データ容量	フォーマットなし 500K バイト	
		データ容量	フォーマットあり 327.68K バイト	256 バイト/ セクタ
		トラック密度	48 tpi	
		シリンダ数	40	

No.	項 目		仕 様	備 考
6	フロッピーディスクドライブ(外付)	トラック数	80	
		記録方式	MFM	
		ディスク回転速度	300rpm	
		データ転送速度	250Kビット/秒	
		回転平均待ち時間	100mS	
		アクセス時間	トラック時間	6mS
			セトリング時間	15mS
		ヘッド・ロード時間	35mS	
		モータ起動時間	400mS	
7	パラレルインタフェース		<ul style="list-style-type: none"> ・セントロニクス(8ビットパラレル)インタフェース準拠 ・ACK 信号応答によるハンドシェイク 	
8	バス・サイクル数	メモリアクセス	T_1, T_2, T_W, T_3 の4サイクル	
		I/O アクセス	$T_1, T_2, T_W(T_W, T_W, T_W) T_3$	()はソフト設定
9	電源電圧および消費電流	電 圧	$5V \pm 5\%$	
			$\pm 12V \pm 5\%$	
		電 流	450mA(TYP)	
10	モ ニ タ		<ul style="list-style-type: none"> ・ROMに内蔵 ・11種のコマンド 	
11	リセット検出電圧		$4.5V \pm 3.3\%$ (PST521C使用)	
12	リセット条件		パワーオンまたは電圧低下によるリセット検出回路動作時	リセットSW付
13	I/O 端子		フラットケーブル用60Pコネクタによる接続 型式:FRC2-C60L11-OL(DDK)	
14	電 源 端 子		4PIN(+5V, +12V, -12V, GND)コネクタによる接続 型式:IL-4P-S3EV2-1(JAE)	

参 考 文 献

- 横田英一：図解マイクロコンピュータ Z-80 の使い方，オーム社（1981）
- HD64180 ハードウェアマニュアル，日立製作所
- COMPONENTS DATA BOOK, Zilog 社
- 各社半導体規格表

索

ア 行

アキュムレータ	23
イミディエイト	33
インデックス	32
インプライド	32
ウェイトコントローラ	8
ウェイトステート	94
エクテンド	33
エッジ入力	166
エリアレジスタ	50
演算命令	36
オートリフレッシュモード	151
オーバフロー	25
オーバランエラー	45
オペコードフェッチサイクル	79

カ 行

カードパソコン	182
擬似 SRAM	151
奇数パリティ	77
キャリー	25
偶数パリティ	77
クロック同期方式	73

引

コモンエリア 0	50
コモンエリア 1	50

サ 行

サイクルスチールモード	135
サイン	25
システム集積型マイコン	4
システムストップモード	126
乗算命令	9
シリアルインタフェース	7
シングルアドレス方式	135
スリープ命令	9
スリープモード	126
ゼロ	25
専用レジスタ	23

ソースアドレスレジスタ	133
-------------	-----

タ 行

タイマ	7
調歩同期式	72
デイジーチェーン	114
デイジーチェーン接続	160
デイジーチェーン方式	11

ディスティネーションアドレス

レジスタ133
デュアルアドレス方式135
転送命令 37

動作モードコントロールレジスタ43
特殊制御命令37
トグル出力 64

ナ 行

ネゲート25
ノンマスカブル割込み102

ハ 行

バイトカウントレジスタ134
バーストモード135
バスリリリースモード124
ハーフキャリー 25
パリティ25
パリティエラー79
バンクエリア50
バンク方式の MMU 7
汎用レジスタ23

物理アドレス49
フラグレジスタ23
フレミングエラー78
プログラム制御命令37
ブロック転送命令143

ベクタ方式104
ベースレジスタ52

ホルトモード122

マ 行

マスカブル割込み102

メモリ空間 34
メモリマネジメントユニット48
メモリライトサイクル40
メモリリードサイクル40

ラ 行

リセット120
リフレッシュコントローラ7
リフレッシュサイクル11, 96
リラティブ33
リロード 64

レジスタインダイレクト32
レジスタダイレクト32

論理アドレス48

ワ 行

割込み方式 11

アルファベット

ACIA82
ASCII72

BIOS 188, 193
BDOS 188
BUSACK 信号 124
BUSREQ 信号 124
BISYNC 73

CP/M80 10

CP/M-Plus 188, 192
 CRTC 167
 CRT コントローラ 167

 DMAC 133
 DMA コントローラ 7

 E 信号 163

 FDC 165

 HALT 命令 122

 I レジスタ 23, 105
 IEF フラグ 107
 IL レジスタ 105
 INO 命令 173
 INT 107
 $\overline{\text{INT}}_0$ 11
 $\overline{\text{INT}}_1$ 11
 $\overline{\text{INT}}_2$ 11
 $\overline{\text{INT}}_0$ モード 0 109
 $\overline{\text{INT}}_0$ モード 1 112
 $\overline{\text{INT}}_0$ モード 2 113
 IO 33
 I/O 空間 34
 I/O 命令 37
 I/O ライトサイクル 43
 I/O リードサイクル 42
 $\overline{\text{IOE}}$ 11
 IOSTP ビット 127
 IX 23
 IY 23

 LCD タイミングコントローラ 167

LCTC 167
 $\overline{\text{LIR}}$ 11

 $\overline{\text{ME}}$ 11
 MLT 命令 9, 173
 MMU コモンバンクエリアレジスタ 50
 MMU コモンベースレジスタ 52
 MMU バンクベースレジスタ 52

 $\overline{\text{NMI}}$ 11, 115

 OUTO 命令 173

 PC 23
 PEO 77

 R カウンタ 23
 RDRF 75
 RE 75, 89
 RET 命令 117
 RETI 命令 117
 RETN 命令 117

 SLP 命令 9, 127, 173
 SP 23

 TDRE 75
 TE 75, 88
 TRAP 116
 TRAP ビット 116
 TST 命令 173

 UART 82
 UFO 116

Z180	18
Z280	18

Z80	7
ZTAT	18

編 者 略 歴

石川 知雄 (いしかわ ともお)

昭和29年 東北大学工学部
通信工学科卒業

昭和35年 スタンフォード大学
大学院修了 (Ph. D.)

現 在 株式会社日立製作所

図解 マイクロコンピュータ

64180の使い方

© 石川 知雄 1988

昭和63年 1月20日 第1版第1刷発行

OHM・OHM・OHM・O
編者承認
検印省略
OHM・OHM・OHM・O
C-KHO・KHO・KHO

編 者 石 川 知 雄

発 行 者 株式会社 オ ー ム 社
代 表 者 種 田 則 一

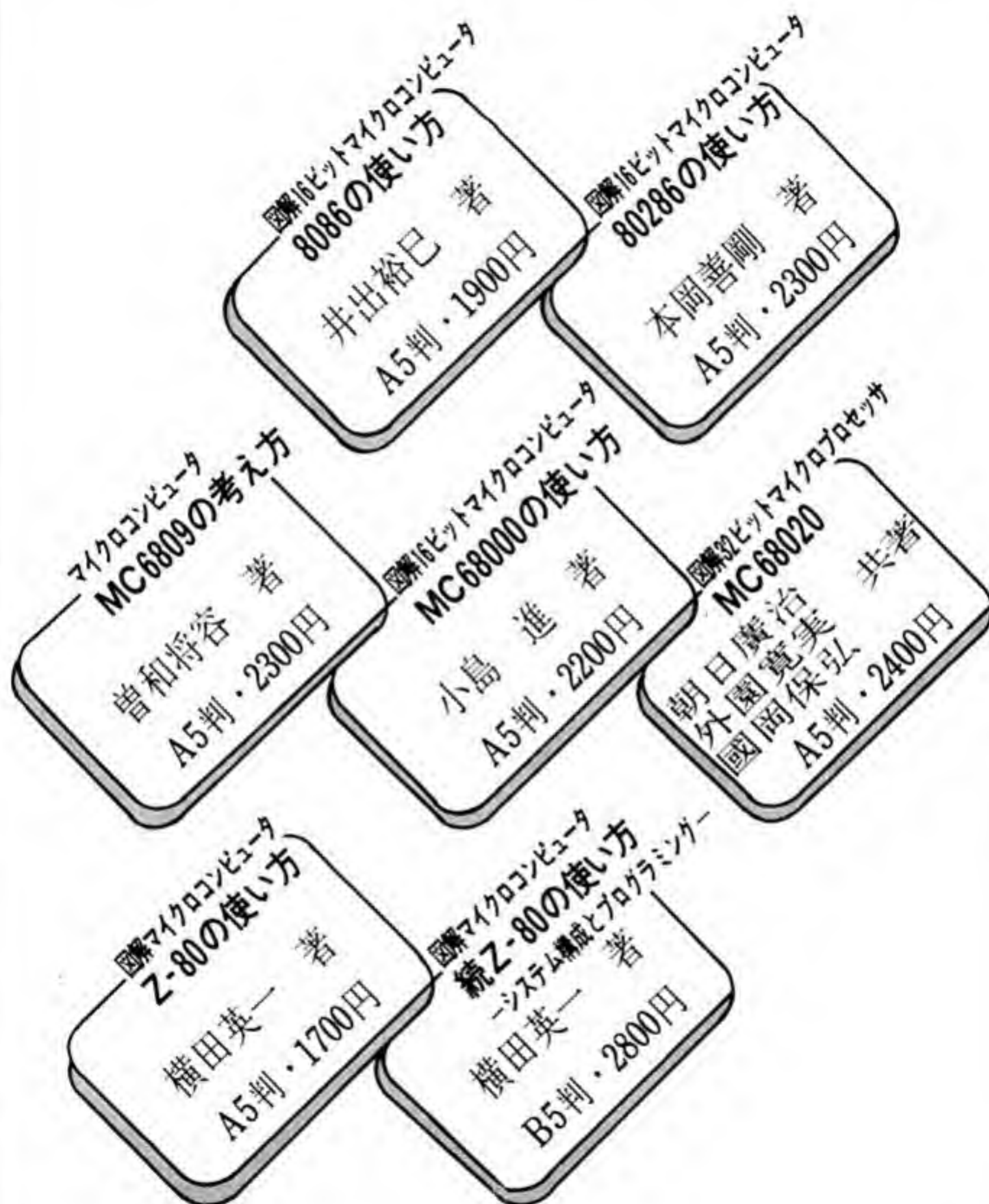
発 行 所 株式会社 オ ー ム 社
郵便番号 101
東京都千代田区神田錦町3-1
振 替 東京 6-20018
電 話 03(233)0641(代表)

Printed in Japan

組版 エヌ・ビー・エス 印刷 日東ブリテック 製本 協栄製本
落丁・乱丁本はお取替えいたします

ISBN 4 - 274 - 07396 - 3

図書案内



— 定価は変更されることがありますのでご了承下さい —



図解コンピュータシリーズ

江村潤朗 監修

コンピュータ・システム入門

江村潤朗・野津 昭 共著 (A5・p.210)

ハードウェア・システム入門

江村潤朗 著 (A5・p.216)

FORTRAN 入門

蛸老沢成享 著 (A5・p.230)

COBOL 入門

蛸老沢成享 著 (A5・p.350)

PL/I 入門

光吉民恵 著 (A5・p.244)

PASCAL プログラミング入門

三沢常男・市川隆男 共著 (A5・p.264)

BASIC プログラミング入門

平山静夫 著 (A5・p.290)

アセンブラプログラミング入門

瀬下孝之・前田忠彦 共著 (B5・p.380)

APL 入門

金子章弘 著 (A5・p.236)

日本語 APL 入門

平尾隆行 著 (A5・p.236)

簡易照会言語入門

—関係データベースシステム—

平尾隆行 著 (A5・p.226)

ストラクチャード・プログラミング入門

國友義久 著 (A5・p.206)

オペレーティング・システム入門

江村潤朗 著 (A5・p.176)

ファイル編成入門

山谷正己 著 (A5・p.184)

データベース入門

穂鷹良介 著 (A5・p.212)

仮想記憶システム入門

山谷正己 著 (A5・p.138)

データ通信システム入門

保坂岩男 著 (A5・p.232)

EDP システム設計入門

南條 優 著 (A5・p.204)

オフィスコンピュータ入門

魚田勝臣・富沢研三 共著 (A5・p.202)

オフィスオートメーション入門

(改訂2版) 中村 茂 著 (A5・p.238)

RPG プログラミング技法

野口正雄 著 (B5・p.232)

プログラム流れ図の作成技法

江村潤朗・野津 昭 共著 (B5・p.358)

対話式計算システムの活用技法

平尾隆行 著 (B5・p.258)

プログラム開発管理

國友義久 著 (B5・p.184)

多重仮想記憶オペレーティングシステム

鎌田 肇 著 (B5・p.216)

コンピュータとアプリケーション

寺沢康夫 著 (B5・p.228)

データベース/データ通信プログラミング

平尾隆行 著 (B5・p.220)

ISBN4-274-07396-3 C3055 ¥2600E

定価 2600 円

